PASCAL - P IMPLEMENTATION ON TDC - 316

A Thesis Submitted
In Partial Fulfilment of the Requirements
for the Degree of
MASTER OF TECHNOLOGY

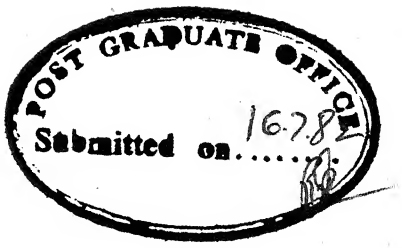
by

S. SRINIVAS RAGHURAM

to the

COMPUTER SCIENCE PROGRAMME
INDIAN INSTITUTE OF TECHNOLOGY, KANPUR

JULY, 1982



CERTIFICATE

Certified that the thesis entitled "PASCAL-P IMPLEMENTATION ON TDC-316" has been carried out under my supervision by Mr. S. SRINIVAS RAGHURAM and has not been submitted elsewhere for a degree.

Kanpur

B. Srinivasan
Computer Science Programme
IIT, KANPUR

dedicated to

the

fond memory of

my

cousin Srinivas

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my guides Dr. B. Srinivasan and Dr. V.M. Malhotra for their constant encouragement and guidance.

I am thankful to Professor H.V. Sahasrabuddhe for the many fruitful discussions I had with him.

I am also grateful to the following :

- George Paul for his DEC to TDC communication software and his package NUT used to format the appendices.
- CHV Murthy for his package DRAFT used to draw the figures in this report.
- Raja for his help in preparing this thesis.
- M/s. Agarwal and Basu for keeping the TDC-316 humming.
- My friends : Aarti Papa, Amar, Anoop, Chattoo, CHU, Kotappa, Kumar, Mahalingams', Prasad, Raja, Raos', Satish, Subi and Tyagi boy for making my stay at IIT Kanpur pleasurable.
- Mr. C.M. Abraham for his excellent typing work,
- and finally to DEC-10, for the many long and challenging hours I spent with it.

Raghuram

ABSTRACT

A single user Pascal programming station based on the Pascal-P compiler has been implemented on a 16-bit minicomputer. The original P-compiler, designed for a 60 bit CDC machine, is modified to handle variable length data types on the 16 bit machine. Run time checks for arrays and subranges have been added. Also implemented are a P-Assembler, a P-Interpreter to interpret the output of the P-Assembler, Pascal file primitives and a command processor. The addition of four relevant instructions and a transparent memory manager and reorganization of the compiler code shows superior compiler performance, based on studies conducted in a simulated environment.

CONTENTS

CHAPTER 1

INTRODUCTION

Portability is a measure of the ease with which a program can be transferred from one environment to another. Portability is important for compiler systems so that we can install the language processor on as many computer systems as possible, in a short time. The trend towards interpreter based systems is on the increase because most of the present day computer architectures are not suited for the easy implementation of advanced high level languages.

Many efforts have been made to design portable programming systems [6,15]. The Pascal-P compiler is a portable compiler for a subset of standard Pascal [12]. The Pascal-P compiler produces object code to run on a P-machine. The P-machine is an idealized stack machine. Code for this machine is termed P-code. The P-code is designed to be compact so that programs in P-code are shorter than the corresponding programs in conventional machine code. This is natural because the instruction set of the P-machine strongly reflects the constructs available in Pascal. The P-compiler is compact because the code generation for the P-machine is simple and straightforward. A P-compiler is highly portable as, it is fairly easy to implement a P-machine. PAS316, which is the subject of this report is a

single user programming system based on the P-compiler and implemented on TDC-316.

The 'P' in P-machine and P-code stands for 'pseudo'. The P-machine may be implemented as a physical processor or may be emulated by an interpreter.

The P-compiler in its original form cannot be efficiently implemented on a 16-bit minicomputer. Hence, it has been modified to handle variable length data types. The instruction set has been modified to accommodate the type information and extended to handle local variables efficiently. The compiler now generates code to check array and subrange bounds.

The P-compiler produces as its object code a P-machine assembly language program. The P-assembler assembles this program and loads it in the P-machines code array, ready for interpretation.

The P-code produced by the P-assembler is interpreted by the P-interpreter. The interpreter also performs machine dependent IO. It fetches each P-code instruction in sequence and performs the appropriate action. The effect of this interpretation is manifested as a change in the state of the P-machines data segment, consisting of the Stack and the Heap.

The environment to run the P-compiler comprises of of about 100K bytes of memory and file-handling primitives.

The P-code contains instructions to handle text files. Pascal file IO operators have been implemented. A temporary filing system has been implemented to handle files. This pseudo-filer is interfaced with the disk through the diskio module. The basic TDC-316 computer supports direct addressing of 64 K bytes of memory. Hardware facilities for memory extension have been used and a transparent memory manager that maps 8 physical segments to 32 virtual segments each of 4 K words has been implemented.

A command processor helps the user to interact with the various subsystems of PAS316. The compilers performance, with the new instructions and the memory manager has been studied in a simulated environment on DEC 10. The segment swap rate has been observed for various test programs.

The poor performance of the P-compiler was attributed to the large number of segment faults. The reason for this is the large number of intersegment procedure calls. An attempt has been made to combine the most often used procedures into contiguous segments, by reorganizing the P-Assembly language program produced by the compiler when compiling itself. There is an appreciable performance improvement after reorganization.

The P-compiler and the various subsystems of the PAS316 system are discussed in the following chapters.

CHAPTER 2

PASCAL-P COMPILER

The Pascal-P compiler is a portable compiler for a subset of standard Pascal. The compiler is written using exactly the subset it processes and is self-compiling. Its output is an assembly language program of a hypothetical stack computer called P-machine. The stack computer is machine independent and also simple. The 'P' compiler was designed such that it can be transported from one computer to another at very little cost in terms of programming effort.

2.1 TDC-PASCAL-P

The Pascal-P compiler released by ETH, Zurich in 1973, is not suitable for implementing on mini-computers. It assumes that all the basic variable types provided in Pascal, to occupy one unit of memory. Thus an integer variable, a real variable, a character variable and a set variable are treated uniformly by the P-code load and store instructions. The initial implementation of the stack computer was on a 60 bit CDC machine. But, on a 16 bit machine like TDC-316, it is unwise to allocate equal store for all types.

Thus, it was necessary to effect modifications to the existing P-compiler at IIT Kanpur, to generate the necessary

type information with the load and store instructions of the stack computer. The P-compiler with the above modifications is implementableon TDC-316.

The P-compiler was not generating code for run-time checking of array bounds and subrange bounds. We strongly felt their necessity. They are now available and can be turned off through the switch $R-.

The P-machine instruction repertoire has been augmented with 4 new instructions. These are :

```
LDLt  Q     ;   load variable at local address Q
STLt  Q     ;   store variable at local address Q
LAL   Q     ;   load local address
UJC         ;   error in case statement
```

't' indicates type, it takes the following values : A,B,I,R and S and denotes address, boolean, integer, real and set respectively. Local implies local to the current activation record. For loading and storing of local variables on the stack, the compiler generates the above new set of local instructions instead of LODt 0,Q, STRt 0,Q and LDA 0,Q respectively.

The handling of the CASE statement has been modified, to indicate error in case of an undefined case label occurring at run-time.

```
VAR I : 0..5;
BEGIN
 CASE I OF
  0 : ;
  2 : ;
  4 :
 END
END.
```

```
L   3
    ENT       L   4
    LDO           8
    UJP       L   5
L   7
    UJP       L   6
L   8
    UJP       L   6
L   9
    UJP       L   6
    L   5
    STR   0       9
I   10
    LOD   0       9
    LDCI          0
    GEQI
    FJP       L   6
    LOD   0       9
    LDCI          4
    LEQI
    FJP       L   6
    LOD   0       9
    LDCI          0
I   20
    SBI
    XJP       L 10
L   10
    UJP       L   7
    UJP       L   6
    UJP       L   8
    UJP       L   6
    UJP       L   9
L   6
    RETP
L   4=            8
```

!code produced by orisinal pcompiler

```
L   3
    ENT       L   4
    LDOI          8
    UJP       L   5
L   7
    UJP       L   6
L   8
    UJP       L   6
L   9
    UJP       L   6
L   5
    STLI          9
I   10
    LDLI          9
    CHKS      0       4
    LDCI          0
    SBI
    XJP       L  10
L   10
    UJP       L   7
    UJC
    UJP       L   8
    UJC
    UJP       L   9
L   6
I   20
    RETP
L   4=            9
```

!code produced by new pcompiler

Fis. 2.1 Effect of Modified CASE statement.

The modification done to the case statement handling can be appreciated by studying Fig. 2.1.

The case bounds are now checked by the single instruction CHKS, and in case of a range error, the execution is aborted. For values of I equal 1 and 3, UJC, instruction is generated. So that, if the values 1 or 3 occur at run-time, the execution is aborted instead of passing it as a no-operation.

2.2 THE PASCAL-P COMPILER OPTIONS

The Pascal-P compiler uses three external files. They are named INPUT, OUTPUT and PRR and are all of type TEXT. The file INPUT contains the source program; the file OUTPUT contains the source program listing (optionally), the symbol tables (optionally), and compiler messages; and the file PRR optionally contains the object program for the P-machine, in its assembly-language form. By 'optionally' we mean that these components may or may not be included in the respective files depending on the user's discretion. These user options can be set at any point in the user program by means of a pseudo-comment. A pseudo-comment is one in which the first symbol within the comment is a '$' symbol. Following the '$' symbol are option settings separated by ',' symbol. An option setting starts with the letters 'L', to indicate 'listing', or 'T', to indicate 'tables', or 'C', to indicate 'code', or 'R' to indicate generation of 'run-time tests' by

the compiler; this letter is followed by either a '+' or a '-' symbol, to turn the option ON or OFF respectively. Default settings for these options are : L is ON; T is OFF; C is ON; R is ON.

2.3 THE PASCAL-P COMPILER PERFORMANCE

The addition of the new instructions has enabled the compiler to produce a high density code. This not only results in saving of memory space, but also reduces the frequency of instruction fetches.

The performance of the compiler with a swappable code space has been studied, in a simulated environment on DEC-10. A memory manager was designed to map 5 physical segments into 14 virtual segments, for the purpose of the study. The replacement policy used is an LRU approximation. [see Chapter 3]. The compiler's speed has been observed to suffer due to a large number of segment faults.

An analysis of the compiler's procedure calling sequence [see Appendix 2] reveals that the number of inter-segment procedure calls are high. A reorganization of the P-code of the compiler was undertaken to cluster the related procedures and reduce inter-segment calls. The reorganization is based on an analysis of the procedure calling sequences of the compiler, for various test programs. A sample of the study results is available in Appendix 2.

The compiler's performance showed an appreciable improvement owing to the reorganization. The memory manager implemented in the PAS316 system, supports resident segments. By using this facility and making the most often used segments resident, the system performance can be improved further. This requires an indepth study of the compiler's procedure calling sequences.

CHAPTER 3

ENVIRONMENT AND SUPPORT FOR PASCAL-P SYSTEM

Nori et al. [13] describe the minimal configuration required to run a P-compiler through an interpreter. These requirements include more than 100K bytes of core memory and availability of a family of high level instructions for transfer of data between memory and external files.

A cursory look at the specifications of TDC-316 is enough to conclude that the machine does neither provide the requisite core nor the necessary file IO operators. Section 3.1 describes the methodology used for creating the high level operators for file handling as required by the PAS316 system. Section 3.2 investigates the various strategies that can be used to circumvent the lack of core memory and the scheme actually implemented.

3.1 ENVIRONMENT FOR PAS316 SYSTEM

The Pascal-P compiler supports the use of four text files. It needs a filing system to handle these files. Depending on the work environment, i.e., interactive or batch, the filing system must be augmented with the necessary IO support.

A Pascal program interacts with the environment only through the four predefined text-files. The PAS316 system

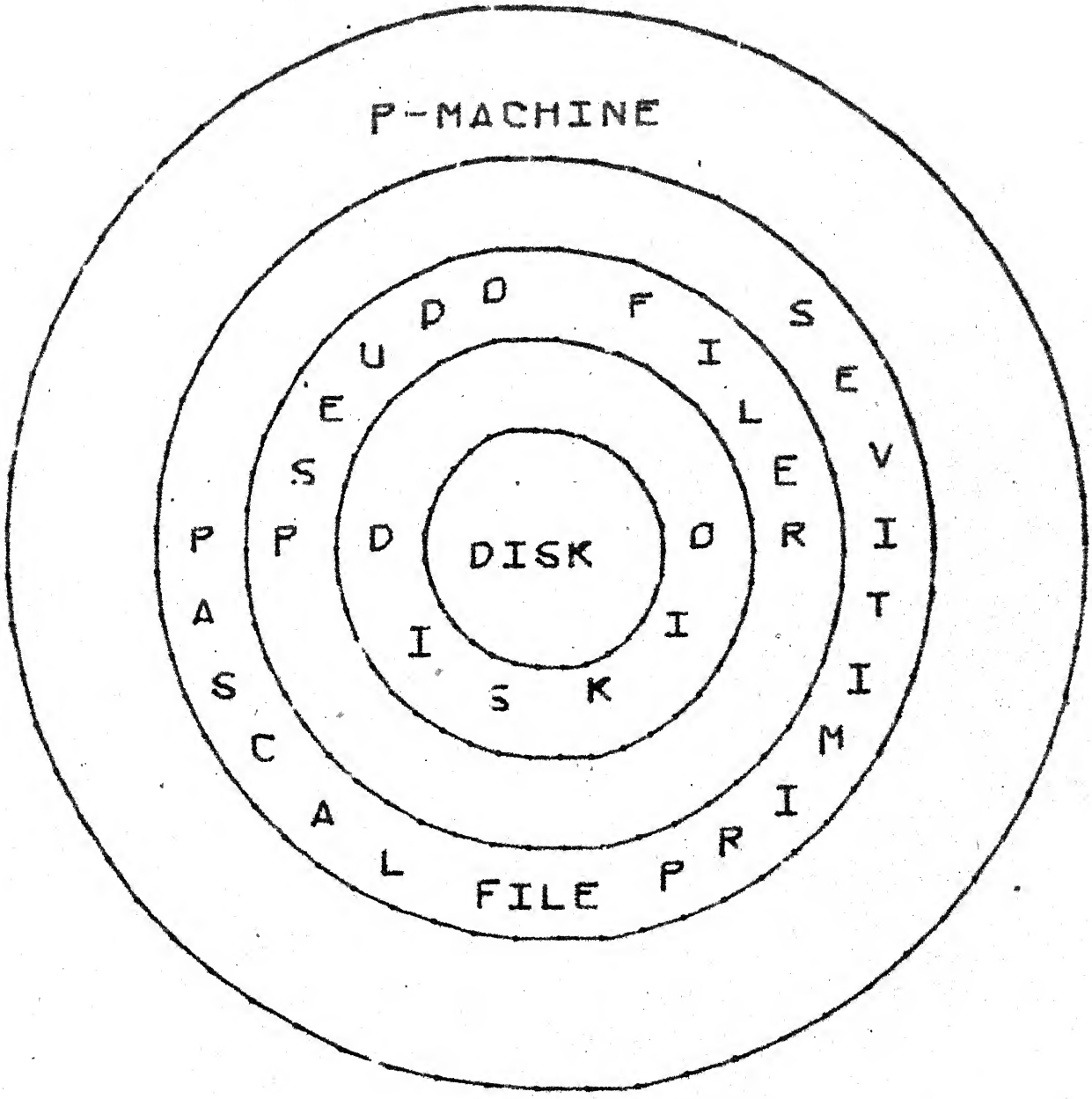


Fig. 3.1 P-Machine's interface with disk

expects all these files to reside on the disk. Since there is no filing system on TDC-316 at present, a temporary interface has been implemented. This interface is designed to support upto six files. In this system the user supplies the starting track, surface, sector address of his files. This parameter termed physical filename, for the purpose of this report may be replaced by the actual filename, when a regular filing system is available.

The PAS316 systems' interface structure with the disk is as shown in Fig. 3.1. A description of this interface follows :

Pascal-P machine as envisaged by Nori et al. accesses the files by means of a set of file IO operators. These operators are provided by the layer PAS316 file IO primitives. This layer enables the P-Machine to manipulate sequential files as if they are the fundamental structure of data handling on the mass-storage device. The transformation between the above view and the reality in which the data is stored as collection of smaller units of information called blocks is provided by the layer pseudo filer. The pseudo filer layer relies on diskio layer for transfer of data between computer memory and disk.

3.1.1 PAS316 - File Primitives [14]

The entire PAS316 system manipulates sequential files using the primitives provided by the module PAS316 file IO.

This uniformity between the P-code instructions operation and the rest of the system could be achieved mainly because of the nonavailability of explicit IO instructions in BLISS11!

A formal definition of a sequence is necessary to understand the working of the elementary file operators and is given below:

(a) $\langle\ \rangle$ denotes the <u>empty sequence</u>

(b) $\langle x_o \rangle$ is called a <u>singleton</u> sequence

(c) If $x = \langle x_1 , \ldots, x_m \rangle$ and

$y = \langle y_1, \ldots, y_n \rangle$ are sequences, then

$x \ \& \ y = \langle x_1, \ldots, x_m, y_1, \ldots, y_n \rangle$

is the <u>concatenation</u> of x and y.

(d) If $x = \langle x_1, \ldots, x_n \rangle$ is a non-empty sequence, then

first (x) $= x_1$

denotes the <u>first</u> element of x.

(e) If $x = \langle x_1, \ldots, x_n \rangle$ is a non-empty sequence, then

rest(x) = $\langle x_2, \ldots, x_n \rangle$

Pascal allows only sequential access to files. The essence of sequential access is that at any time only a single specific component of the sequence is immediately accessible. This component is specified by the current position of the access mechanism. This position can be

altered using file operators. The most frequently used operation is to move to the next component of the file. Another operation of interest is to move the current pointer to the beginning of the entire sequence.

Also, a file can only be in one of two states : either in the state of being constructed (written) or of being scanned (read). We can formally define the position of current pointer of a file by regarding a file x to be partitioned into two parts, part x_L to the left of this pointer and a part x_R to its right. Hence,

$$x = x_L \; \& \; x_R \quad \text{(concatenation)}$$

Associated with each Pascal file is an implicit auxiliary variable that represents a buffer. This variable has the same type as the base type of the file. Since Pascal-P supports only text files, all buffer variables are of type character. The buffer variable of a file x is denoted by $x\uparrow$.

Pascal provides the following elementary file operators to its users.

i) Constructing the empty tfile.

rewrite (x) = x:= $\langle\rangle$

The operator rewrite is used to erase any existing file x and initiate the process of constructing a new file, with name x.

ii) Extending a file

Put(x) = x:= x & x↑

The operator put appends the value of x to the file x.

iii) Initiation of a scan

reset(x) = x_L:= ⟨ ⟩ ;

x_R:= x ;

x := first(x)

The operator reset is used to initiate the process of reading a file. It sets the current pointer for the file to its first component.

iv) Proceeding to the next component

get(x) = get(x_L & x_R) = x_L:= x_L & first(x_R) ;

x_R:= rest(x_R);

x := first(rest(x_R)).

note : first(s) is defined only if s ≠ ⟨ ⟩ .

v) The predicate

eof(x) = x_R = ⟨ ⟩

The operation get(x) can therefore only be executed if the predicate eof(x) is false.

To ease the task of handling text files, the Pascal user is also provided with the following additional operators.

vi) read(x,VAR,v) = v:= x↑ ; get(x)

prerequisite: not eof(x)

vii) Write (x,e); e is an expression of type character
= x↑ :=e; put(x)

prerequisite: eof(x)

The line marker used is the ASCII character sequence CR LF.

viii) Writeln(x) append a line marker to the file x.

ix) Readln(x) skip characters on file x upto the one immediately following the next line marker.

x) Eoln(x).

A boolean function. True, if the file position had been advanced to a line marker; false otherwise. Also, if eoln(x) is true then x↑ = blank(ASCII 40B).

We shall now investigate the functions provided by each of the layers shown in Fig. 3.1.

3.1.2 PAS316 - File IO

This module enables the programmer to perform the elementary file operations as defined in the previous section. The user interacts with his files through a logical filename. For the purpose of this implementation the logical filename is required to be an integer between 0 and 5. The physical filename now consists of three integers denoting

the track, surface and sector address of the file on disk. This parameter can be replaced by a physical filename when a regular filing system is operational.

The user has to supply a one character buffer, for each of his files. This constitutes the auxiliary variable x↑ defined in Section 3.1.1.

The main data structure manipulated by the file operators is

```
filetable : ARRAY[0..5]  OF (* six files *)
                RECORD
                  fsp: BYTE; (* file specifier returned
                                by filer *)
                  eoln: BOOLEAN;
                  eof : BOOLEAN;
                auxch  : BYTE; (* needed for managing
                                  EOLN lookahead *)
                bufadr : WORD (* user supplied one
                                 character buffer *)
              END;
```

This module interacts with the pseudofiler described in Section 3.1.3.

The entry points of this module are :

Legend : l = logical filename;
b = Buffer address;
p = physical filename.

a) reset (l,b,p);

b) rewrite (l,b,p);

c) get (l);

d) put(l);

e) readln (l);

f) writeln (l);

g) read (l,VAR ch);

h) write (l,ch);

i) readint (l,VAR N);

j) endofline (l);

k) endoffile (l);

l) fileclose (l);

For details refer the program listing provided in Appendix 6.

3.1.3 Pseudo-filer

The pseudo-filer is nothing but a buffer manager, handling six buffers. It has an array of six buffers. A buffer is allocated to a file when it is opened and deallocated when a file is closed.

The data structures handled are :

```
buffertable : ARRAY[0..5, 0..255] OF BYTE;
openfilestable: ARRAY[0..5] OF
                    RECORD
                    baseadr: WORD; (* base disk address *)
                    curadr : WORD; (* current disk address *)
                    purpose: BYTE; (* read/write)*)
                    bufferptr: BYTE (* current byte pointer *)
                  END;
```

A file can be opened for either reading or writing, by giving the physical file name. The module returns a file specifier (fsp) for each file opened. This file identifier can be used for accessing the file by the user.

The entry points of the pseudo-filer are :

a) open (physical filename, purpose, VAR fsp, VAR flag);
   If flag = true is returned implies buffers exhausted.

b) Close (fsp);

c) getbyte (fsp, VAR ch);

d) putbyte (fsp, ch);

3.1.4 Disk - IO

The pseudo-filer uses the disk-IO to put and get sectors from the disk.

This module is totally device dependent and interacts with the Disk controller of TDC-316 to transfer a given number

of words between the disk and memory.

The routines that perform the transfer are :

a) putblk (track, surface, sector, word count, memory address)

b) getblk (track,surface,sector,word count,memory address);

3.2 MEMORY MANAGEMENT

Clearly the memory requirements of a Pascal-P system are too large to implement on the basic TDC-316 computer. Some form of overlay technique is essential. There are a number of ways of tackling this problem.

3.2.1 Multi-pass Compiler

Writing a multi-pass Pascal Compiler ! This as is obvious will call for major changes in the design philosophy of the existing Pascal-P compiler. Further, it will cost us the elegance and simplicity of the P-compiler.

3.2.2 Code Segmentation

An alternative solution is to use virtual memory techniques using Pascal procedures as virtual memory segments. This approach is attractive as it avoids the error prone alteration of the compiler code!

In this scheme, the P-code can be split into segments by the compiler, each segment consisting of the code of one procedure. Segmentation of the data segments would call for considerable effort.

Hence, a proposal to restrict the virtual memory operations to the pure code block of the P-system and to use in core memory for the other segments has the following advantages:

a) Since the swappable segments contain pure code they need not be copied on the backing store on swapping.

b) Since a goto statement leading out of a procedure body is not available in the Pascal-P language, access to a segment occurs only upon procedure entry and exit. Hence, segment management considerations have to be taken care only on the CALL and RETURN P-instructions.

c) No interference to data accessing and addressing is necessary, which normally is cause for excessive overheads, if performed through software.

This proposition is attractive for machines, lacking in hardware facilities for memory management, and suffering from lack of large address space. (e.g. 8080AμP). We, therefore, analyse this scheme in greater detail.

The memory management is likely to be complex as procedures vary in size. This also leads to external fragmentation. Further, as the replacement policy has to be implemented exclusively through software, it will involve considerable overheads. Thus, leading to considerable slackening of an already slow interpreter.

For a machine with a main memory of 64K bytes, after satisfying the memory needs of the various resident segments, the swap space available will be relatively small. We predict severe thrashing in such environments. This problem is further compounded by the structure of the compiler.

Owing to the recursive descent nature of the compiler, the number of active procedures is generally large. In addition, the most often used procedures like lexical analyzer and expression parsing routines are comparatively large.

Example 3.1 : (Failure of strict LRU policy)

If we use an LRU replacement policy, because of the lack of swap space and due to the considerable variation in the size of the segments [see Appendix 2], we often have the problem of deleting a number of segments to bring in one segment.

Let the following table indicate the segments present in memory in order of decreasing age.

Segment	Size in units/blocks of memory
S0	4
S1	6
S2	1
S3	15

Table 3.1

```
PROCEDURE expression;
  PROCEDURE simpleexpression;
    PROCEDURE term;
      PROCEDURE factor;
        BEGIN
        .
        .
        END; (* factor *)
      BEGIN (* term *)
        factor;
        WHILE sy = mulop DO
          BEGIN
            .
            .
            factor;
            .
            .
          END
      END; (* term *)
    BEGIN (* simpleexpression *)
      .
      .
      term;
      .
      .
      WHILE sym = addop DO
        BEGIN
          .
          .
          term;
          .
          .
        END
      END; (* simpleexpression *)
    BEGIN (* expression *)
      simpleexpression;
      IF sy = relop THEN
        BEGIN
          .
          .
          simpleexpression;
          .
          .
        END
    END; (* expression *)
```

Fig. 3.2 Structure of Procedure Expression

If now a request to swap-in a segment of size 14 units is issued. All the 4 segments get removed.

To circumvent the problem dealt in Example 3.1, one may be tempted to device a strategy that deletes the largest segment of the eligible segments. This policy is also not sound enough.

Example 3.2 (Thrashing during expression parsing)

Assume the expression being parsed is

A + B

The call traffic to parse the factor A is as shown in Table 3.2. The structure of the procedure expression is summarised in Fig. 3.2. Each procedure in the expression block is relatively large.

Swapped out	Swapped in Expression	Call
Expression	Simple expression	Call
Simple expression	Term	Call
Term	Factor	Call
Factor	Term	Return
Term	Simple expression	Return
Simple expression	Expression	Return

Table 3.2

```
PROCEDURE insymbol;
  BEGIN (* insymbol *)
    CASE ch OF
      (*letters*).:BEGIN
             (* code for identifiers and reserved words *)
                  END;
      (*digits*)..:BEGIN
             (* code for integer and real constants *)
                  END;
       ''''       :BEGIN
             (* code for string constants *)
                  END;
       (*etc*)
     END (* case *)
   END; (* insymbol *)
```

Fig. 3.3 Structure of Procedure Insymbol showing mutually exclusive code

```
PROCEDURE insymbol;
  PROCEDURE inident;
    BEGIN
   (* code for scanning identifiers and reserved words *)
    END;
  PROCEDURE innumber;
    BEGIN
   (* code for scanning number denotations *)
    END;
      .
      .
  BEGIN (* insymbol *)
    CASE ch OF
      (* letters *)...:inident;
       (* numbers *)...:innumber;
      .
      .
    END; (* case *)
  END; (* insymbol *)
```

Fig 3.4 : Restructured procedure Insymbol

```
PROCEDURE expression(...);
  PROCEDURE simpleexpression(...);
    PROCEDURE term(...);
      PROCEDURE factor(...);
        PROCEDURE facident(...);
          BEGIN
            (* code to handle identifiers *)
          END;
        PROCEDURE facset;
          BEGIN
            (* code to handle sets *)
          END;
        BEGIN (* factor *)
          .
          CASE sy OF
            ident : facident;
            strinsconst : facstrins;
            lbrack : facset;
            .
          END; (* case *)
          .
        END; (* factor *)
      PROCEDURE mulfac;
        BEGIN
          (* handle <mulop><factor> *)
        END;
      BEGIN (* term *)
        factor(...);
        WHILE sy = mulop DO mulfac;
      END; (* term *)
    PROCEDURE addterm;
      BEGIN
        (* handle <addop><term> *)
      END;
    BEGIN (* simpleexpression *)
      .
      term(...);
      .
      WHILE sy = addop DO addterm;
    END; (* simpleexpression *)
    PROCEDURE relsexpr;
      BEGIN
        (* handle <relop><simpleexpression> *)
      END;
  BEGIN (* expression *)
    simpleexpression(...);
    IF sy = relop THEN relsexpr;
  END; (* expression *)
```

Fig. 3.5 Restructured Procedure Expression

It was observed that large parts of code are loaded and after execution of only a few instructions get swapped out.

Thus, some efficiency can be obtained by breaking the larger procedures (especially most often used, see Appendix 2) into small procedures. Two obvious candidates for this operation are Insymbol and Expression block. A simple and effective strategy for partitioning programs was suggested by Hurst [10].

This strategy is based on the observation that the procedures usually have mutually exclusive sets of code. Fig. 3.3 illustrates sets of mutually exclusive sections of the procedure insymbol. Hurst suggests that these mutually exclusive sections be used to define new procedures. Fig. 3.4 and Fig. 3.5 respectively illustrate the application of this strategy on procedures insymbol and expression.

Example 3.3 :

The effect of applying the above sub-division was studied on the procedure Insymbol. During the compilation of the binary tree traversal program shown in Appendix 2, it has been observed that procedure Insymbol was called 398 times and procedure Innumber was not called at all!

3.3 PAS316 MEMORY MANAGEMENT

We for the purpose of this project have not pursued the

strategy of Section 3.2.2, because of its excessive cost. We instead, decided to exploit the additional hardware features of TDC-316 to build a rudimentary virtual memory manager. To this end we describe the relevant hardware on TDC-316, in the next Section.

3.3.1 TDC-316's Memory Allocation and Protection (MAP) [7]

The TDC-316 MAP unit provides hardware facilities for memory management for systems where the system memory size is greater than 28K words.

The basic characteristics of the MAP unit are :

i) 16 user mode memory segments
ii) 16 superviser mode memory segments
iii) 8 segments each for instructions and data in each mode
iv) Segment lengths from 32 to 4096 words.

Eventhough, the wordlength and operational logic of TDC-316 is of 16-bit length, the CPU and Bus addressing logic are actually of 18-bits. These extra two bits provide the basic framework for expanded memory operation.

The MAP unit makes available in the supervisor and user mode, 56K words of virtual address space (28K of Instruction and 28K of data). However, this would require that the programs resident in the I-space be pure. If not, D-space cannot be enabled and only 28K words of virtual space will be available in each mode.

Even this extended memory facility cannot directly be used for the Pascal-P system. Pascal-P needs data space in excess of what is available. The P-code interpreter regards both the P-code and the data segment as data. Thus eventhough, an addressing capability of 56K words is adequate, 28K words of data space is inadequate.

We decided to use the MAP unit and provide the required data space by using the Transfer Communication Mode instructions of TDC-316. This enabled us to use the hardware support available for memory management.

This decision costs us execution of 2 additional instructions for each fetch cycle of the P-machine (i.e. for each access to the P-machine's code array). The scheme implemented is shown in Fig. 3.6.

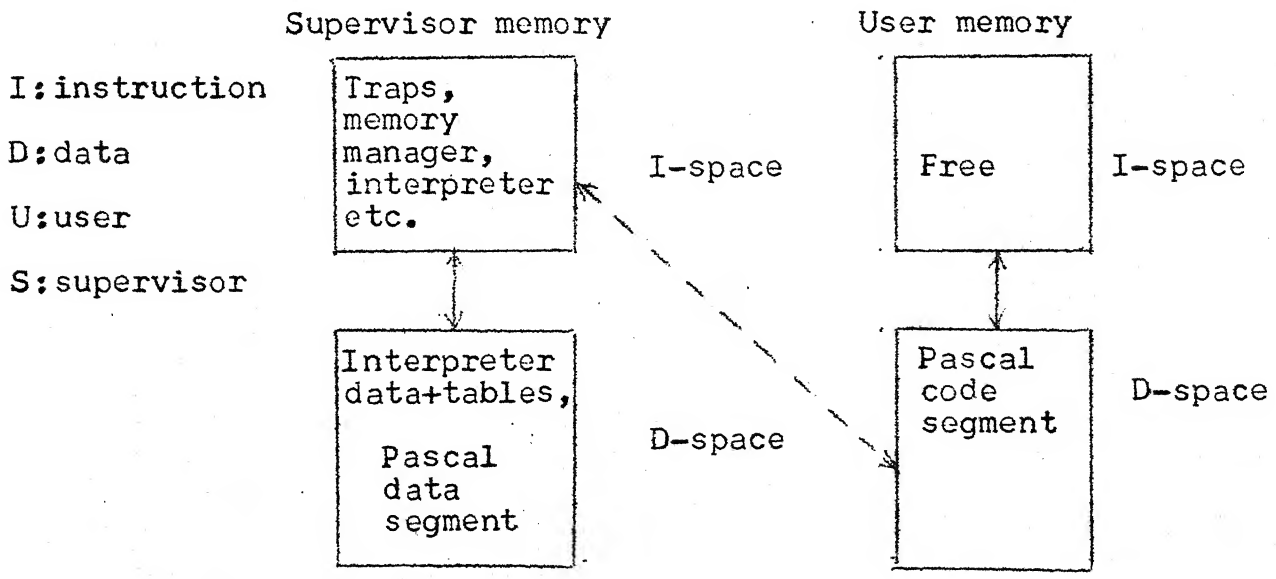


Fig. 3.6 PAS316 Memory Allocation Scheme

```
PROCEDURE      fetch (VAR   v);
  BEGIN
    transfer communication mode to FROM UD space;
    v:= pcode [ppc];
    transfer communication mode to NORMAL
  END; (* fetch *)
PROCEDURE deposit (v);
  BEGIN
    transfer communication mode to TO UD space;
    pcode [ppc]:= v;
    transfer communication mode to NORMAL
  END; (* deposit *)
```

3.3.2 Segmentation

The 32K words of total addressable area in each of the 4 spaces, is divided into 8 segments. The segments can vary in length from 32 words to 4K words in multiples of 32 words. Thus, there are a total of 32 segments.

Each of the 32 segments has a relocation constant (MAP-AF), a length constant (MAP-LF) and an access control specifier (ACF) associated with it. These are stored in a register set called Active Segment Register (ASR), file.

3.3.3 MAP Operation

When the MAP unit is enabled, all addresses generated by the CPU are routed through it. The current processor

status specifies one of the four spaces, while the segment selection is through the three most significant bits of this virtual address. The relocation constant is combined with the 16-bit virtual address to get a relocated 18-bit physical address. At the same time the segment length and the access type are checked against the length constant and access control specifier for validity. If the access is a valid one, the relocated address is put on the bus for data access. Otherwise, the relevant information is strobed into the MAP status registers and access is aborted and a trap is generated. The status registers contain relevant information for program recovery and roll back.

3.3.4 Segment Manager

Since the Pascal-P code and Data Segments are contiguous arrays the maximal allowed segment size of 4K words was chosen.

Thus the memory manager has to handle the 32 virtual segments in 4 spaces, through swapping in the 8 pages of core available on TDC-316. As the 8th segment in each space is required to be set aside for register and ROM use, we essentially have 28 virtual segments and 7 physical pages each of 4K words.

The memory manager has a fixed 128K word swap space on the disk to accomodate the 32 virtual segments. The swap space has been organized such that the disk address calculation

is simple and once the transfer of data begins, there is no additional seek delay. The swap space has been organized as follows :

The swap space for each segment is on a separate track. Hence, mapping from virtual segment number to track address is one to one. In each track four surfaces are used, each surface stores 1K words. The eight sectors needed to store 1K words have been chosen such that, all the 1K words can be transferred at a time through DMA, and when the surface is switched the free sectors pass under the head. Thus, even latency delays are minimized.

3.3.5 Replacement Policy : Passing the Parcel [12]

The segment replacement is based on an LRU approximation. It can best be described as Not Used Recently (NUR) policy.

The policy can be best understood by visualizing the segments present in core to be involved in the passing the parcel game. At any instant of time a segment which is in core will be holding the parcel. A segment holding the parcel, is considered as the next candidate for swap out by the manager. It is actually marked OUTCORE, even though is in-core.

When a trap occurs (music stops!), if the reference is to the parcel holder, then it passes the parcel and is marked, INCORE. This case is called a dummy trap, because no swapping of segments has actually taken place. If the reference was

not to the parcel holder, it is swapped out and the necessary segment is brought in.

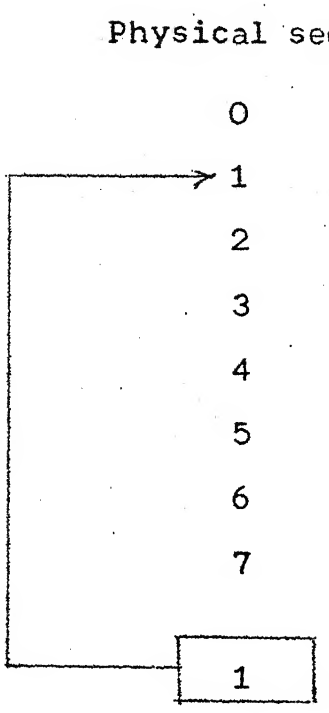


Fig. 3.7 Core table

```
ALGORITHM  swappolicy;
  VAR
    coretable : ARRAY[0..7]  OF (*8physical segments *)
                RECORD
                segno      0..32; (* 32 virtual segments *)
                otherinfo
                END;
    parcelholder : 0..7;
```

```
PROCEDURE  passparcel;
  BEGIN
    give parcel to the nearest eligible neighbour;
    (* round Robin *)
    (* a facility to make some segments permanently resident
    is available.  These segments never hold the parcel *)
    make coretable [parcelholder]·segno OUTCORE
  END; (* passparcel *)
PROCEDURE   initpolicy;
  BEGIN
    FOR I:= 0 TO 7 DO
      Coretable [I]·segno:= 32; (* invalid segno *)
    parcelholder:= -1; passparcel
  END; (* initpolicy *)
PROCEDURE  system-wants (seg);
  (* segment fault has occurred. System needs the segment
  SEG *)
  BEGIN
    IF coretable [parcelholder]·segno ≠  seg  THEN
       BEGIN
         Swapout (parcelholder);
           Swapin(seg,parcelholder)
       END;
    make coretable [parcelholder]·segno INCORE;
    passparcel
  END; (* system-wants *)
```

3.4 CONCLUSIONS

We have described the strategies used to provide the requisite environment for running a Pascal-P system.

In Appendix 2, we show the performance of the memory manager. The relation between the trap rate and the procedure calling sequence has been studied.

The Pascal file operators implemented have been used uniformly throughout the implementation. These were used as the IO primitives for the rest of the system.

CHAPTER 4

P-ASSEMBLER AND P-INTERPRETER

The P-compiler transforms a Pascal program into an equivalent P-Machine's assembly language program. The P-Assembler assembles this assembly language program. The syntax of the P-Machine's assembly language is given in Appendix 4. The P-Assembler loads the P-Machine's code array PCODE with the object code, ready for interpretation. The object code can be salvaged into an object file by using the PSAVE program.

The P-interpreter interprets the object code available in the code array PCODE. It can accept input from two standard files INPUT and PRD and can generate results in the two standard files OUTPUT and PRR.

4.1 P-ASSEMBLER

The P-assembler is a single pass assembler with back patching. It accepts the source program in file PRR and loads the object code in the P-machine's code array PCODE. It uses a mnemonic table and a labeltable. The label table is used for storing the addresses of tables used in the assembly language and for back patching.

4.1.1 Data Structures

```
TYPE
  labelrange = 0.. max-no-of-labels
  instrange  = 0 .. max-no-instr;
  spfcnrange = 0 .. max-no-spfcns;
  labst = (entered,defined); (* label status *)
  alfa  = PACKED ARRAY [1..4] OF CHAR;
VAR
  inst-table : ARRAY[instrange] OF alfa;
  (* binary search performed to locate mnemonic *)
  special-fcnproc-table : ARRAY[spfcnrange] OF alfa;
  (* linear search to locate function/procedure mnemonic *)
  opcode-table: ARRAY[instrange] OF INTEGER.
  (* opcode stored,correspond to mnemonics in instruction
  table *)
  (* label table *)
  labelvalue: ARRAY[labelrange] OF INTEGER;
  label-status: ARRAY[labelrange] OF labst;
  Pcode: ARRAY[0 .. maxcode] OF BYTE;
```

4.1.2 Algorithm

The generation of the program from the BNF definition of the Assembly language defined in Appendix 4 is relatively straight forward. Since the number of instructions of opcode class 1 are relatively large, the P-assembler resolves the opcode class using a CASE statement rather than storing this information in the opcode table.

```
ALGORITHM    P-assembler;
  BEGIN
    init;
    generate;
    ppc:= 0;
    generate
  END; (* P-assembler *)
```

Generate produces the object code for an assembly record.

```
PROCEDURE generate;
  BEGIN
    WHILE NOT eoln (source-file) DO
      BEGIN
        process-assembly-statement;
        Readln (source-file)
      END
  END; (* generate *)
PROCEDURE process-assembly-statement;
  BEGIN
    CASE first character of line OF
      'I' : ignore the line;
      'L' : update-label-table;
      ' ' : assemble-the- instruction
    END
  END; (* process-assembly-statement *)
```

```
PROCEDURE update-label-table;
  BEGIN
    i) get the label value, which is either the current value
       of location counter (ppc) or is as defined in this
       source line;
    ii)If there was a forward reference of this label then
       traverse the code array and update;
   iii)Store value of label in the label table
  END; (* update-label-table *)
PROCEDURE assemble-the-instruction;
  BEGIN
    getmnemonic;
    binary search instruction table and get
      corresponding opcode value from opcode table;
    CASE opcode class of instruction OF
      procedure call instruction                : ... ;
      return instruction                        : ... ;
      load constants                            : ... ;
      special instr. CHK                        : ... ;
      special instr. MST                        : ... ;
      opcode class 2                            : ... ;
      opcode class 3                            : ... ;
      opcode class 4, ENT instruction           : ... ;
      opcode class 5                            : ... ;
      opcode class 6                            : ... ;
      opcode class 7                            : ... ;
      opcode class 8                            : ... ;
      others (opcode class 1)                   : ... ;
  END
```

The transformation from the assembly language to the P- machine's instructions given in Appendix 5 is straightforward except in the following cases :

Assembly Instruction	Code generated
LDCB 0	LDCBF
LDCB 1	LDCBT
LDCN nilvalue	LDCN
CSP special fcn name	Specialfcn name
Eg: CSP WRI	WRI

4.2 P-INTERPRETER

4.2.1 P-Machine's Architecture

The P-machine consists of 7 registers and a memory. The registers are

PPC - P-machine's program counter

PSP - P-machine's stack pointer

MP - Mark pointer, points to the base of the current activation record

NP - New pointer, points to the top of the Heap

OP - opcode

P - P-operand

Q - Q-operand.

OP,P and Q together from the Instruction Register.

The memory is organized as two linear arrays. One called PCODE and the other PSTK. The PCODE is an array of bytes which is accessed through the two procedures fetch and deposit. The PSTK is an array of words. PPC is an index into PCODE and PSP, MP and NP are indices into PSTK.

The Stack and the Heap as required by the executing Pascal program grow towards each other starting at the two ends of PSTK.

The stack grows from 0 upwards and consists of all directly addressable data according to the data declaration. Storage overflow occurs if PSP and NP meet. The Heap grows downwards from the top of the PSTK.

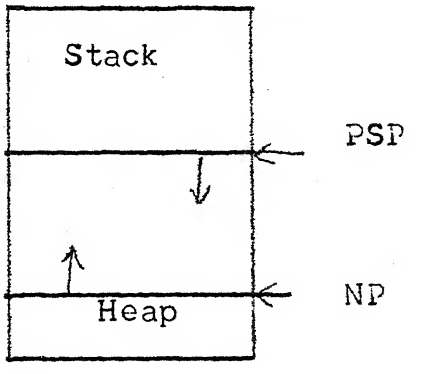


Fig. 4.1 PSTK

The Heap grows when the memory allocation is done through the standard procedure NEW.

The Stack consists of a sequence of activation records, each belonging to an active procedure or function. The activation record starting at PSTK [0] belongs to the main block of the Pascal program.

An activation record consists of the following sequence of information.

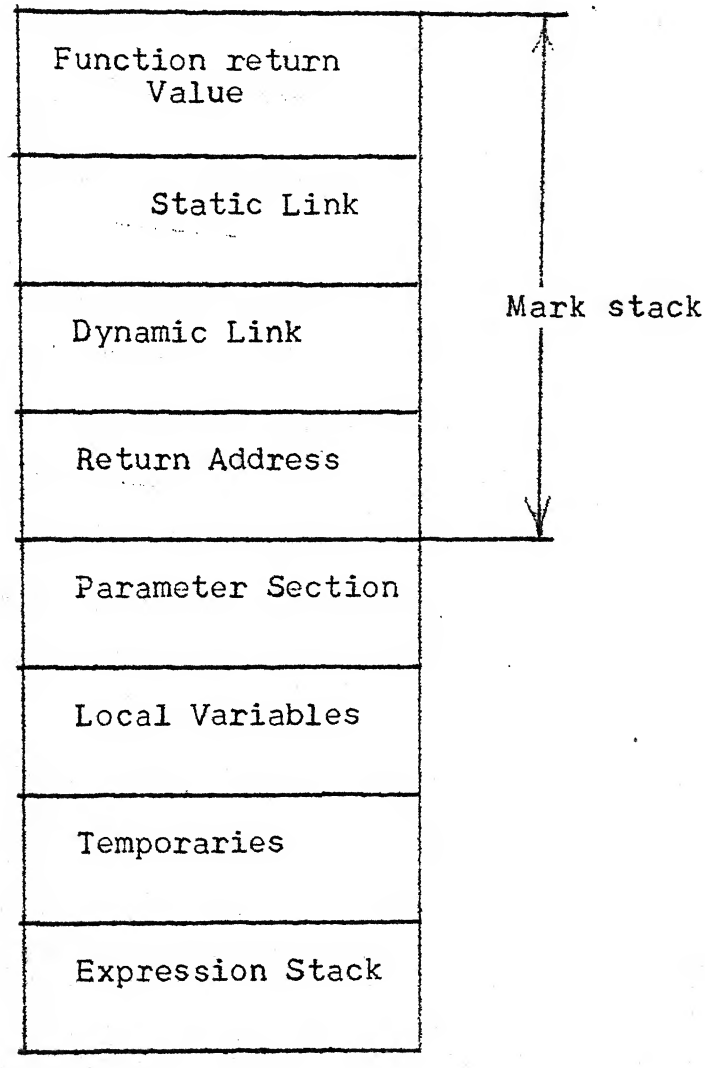


Fig. 4.2 Activation Record

a mark stack part; a parameter section if there are any parameters to the procedure or function to which the activation record belongs; a local-variables sections if there are any local variables declared within the procedure or function to which the activation record belongs; any temporary elements which may be required in the program-evaluation process.

The mark stack paft has further internal structure. It is 5 words long. The first field is space for preserving the value returned by a function. It is included even for procedures for the sake of uniformity. This field is of 2 words, to enable returning real values. The second field is a pointer called static link; the third field is a pointer called dynamic link; the fourth field is a pointer called the Return address. The static and dynamic links are indices into PSTK and return address is an index into PCODE.

There are two types of parameters :

a) pointers (into PSTK) in case the corresponding parameters are of type call by reference or of type call by value but of record or array type or

b) the parameter is call by value and the value is passed.

Before a procedure or function call is made, a mark stack instruction (MST) is executed with a parameter which allows the links to be filled.

Then a series of expression evaluations fill in the parameter section. After this a call user procedure (CUP) or a call standard procedure (CSP) instruction is executed.

4.2.2 P-Machine's Instruction Set and Addressing Modes

The P-Machine's Instruction set is given in detail in Appendix 5. The constructs defined in Pascal are strongly reflected in the instructions of the P-machine.

A set of 4 new instructions have been introduced into the P-machine, to supplement those defined in the original P-compiler [13]. These are LDL, STL,,LAL and UJC.

The introduction of new instructions to load and store local variables is based on the premise that variables local to a procedure are accessed more frequently and therefore require fast access by short instructions, whereas access to remote variables is relatively rare and requires less efficiency.

The LDL, STL and LAL are new mnemonics for the original P instructions LOD 0, Q; STR 0, Q; and LAL 0, Q with the parameter 0 dropped.

UJC, is used to trap undefined case labels at run time. The compiler generates the UJC instruction for all values of the case variable, that have not been defined but lie within the range of the case statement (Refer Fig. 2.1.)

Hence, access to the most often used variables, which in general are the local variables and global variables (variables defined in the outermost block of the Pascal program) are accessed through base registers. The base of the current activation record is stored in the register MP and the base of the outermost block is 0.

Instructions consist of one or several bytes. They can be divided into four basic categories : load and store instructions, operators, control instructions and miscellaneous instructions :

The load and store instructions transfer data between memory (stack or heap) and the top of the stack, where they are accessed by operators. The top of the stack, where data are loaded as intermediate results is also called the expression stack.

The system allows the creation of an activation record or a dynamic variable, only if there are atleast 100 words available, for the expression stack. This feature is based on the premise that the amount of stack space needed for evaluation of an expression is less than 100 words, which is generally true. Load and store instructions involving local and global variables require only a single offset address because the stack address is implicit. For intermediate variables, an additional parameter specifies the level difference, which indicates, the number of levels to descend

using the static links for accessing the variables. The load and store instructions are further subdivided depending on the data size. The transferred data may be a word (16 bits), a double word or four words.

The addressing modes available on the P-machine are defined as follows :

(p and q denote the instruction parameters, 'a' denotes the resulting address and psp is stack pointer) :

- local mode : a = MP+q, used for variables local to procedures.
- Global mode : a = q, used for global variables.
- Stack mode: a = base (p) + q, used for intermediate variables.

  Base (p) returns the contents of the dynamic link field of the activation record which is p levels below the current activation record. Base descends the stack using the static link fields of activation records.
- Indirect mode : a = pstk [psp] + q, mode used for indirect addressing and access via pointers.
- Indexed modes : a = s1 + q * s2, s1 is the arrays base address, and s2 the computed index (s1, s2 are on stack at pstk [psp-1] and pstk[psp]), q is a multiplier depending on the size of the accessed data type.

- immediate mode: the immediate mode is used to load constants. The size of the operands depends on the type of the constant being loaded. The operand size may be 0,1,2, 4 or many words depending on the constant being loaded is a boolean constant or nil, an integer, a real, a set or a string respectively.

The second category of instructions are the operators. They take operands from the top of the stack and replace them by the result. The P-machine instruction set includes operators for integer, real, boolean and set arithmetic. There is a full set of comparison instructions which generate a boolean result. Distinct sets are available for integer, real, boolean and set comparisons.

Control instructions include procedure calls, procedure returns and jumps. Conditional jumps are generated for IF, WHILE, REPEAT and FOR statements. Indexed jumps are generated for CASE statements. They fetch their operand from the stack. There are explicit instructions for checking the array index bounds and subrange bounds, and undefined case labels.

The miscellaneous instructions category contains operators for reading and writing data on files, marking stack, entering a block and dynamic memory allocation and deallocation.

4.2.3 P-Instruction Execution

For the purposes of execution, the opcodes are divided into three categories. The type of the opcode is detected by the interpreter by checking one bit of the opcode. The P-interpreter program is available in Appendix 6. The algorithms used for interpreting the various instructions are self-explanatory.

```
ALGORITHM  P-interpreter;
  BEGIN
    interpret: = true;
    WHILE interpret DO
      BEGIN
        fetchopcode;
        CASE type-of-opcode OF
        type 0 : execute;
        type 1 : fetchoperands; execute;
        type 2 : fetchoperands; set registers execute
      END
    END (* interpret is made false when a STP instruction
        is executed *)
END ; (* P-interpreter *)
```

CHAPTER 5

CONCLUSION

The P-Assembler, P-interpreter and the support software have been written and implemented on TDC-316. Except for a small module, all modules were written in BLISS11. The remaining module concerned with restoring of computer registers after a memory management trap, was coded in assembly language, to have explicit control on the code generated.

The developed software has not yet gone through exhaustive performance evaluation because of the constraints of time. But the limited experience we have is satisfying.

A user interacts with the system through the command processor. He can use this processor to compile, assemble and interpret his Pascal program. If he so desires he can direct the command processor to skip any one or more of the above steps. An user manual is available in Appendix 8. Fig. 5.1 indicates the structure of the PAS316 system.

5.1 CRITICAL REVIEW OF PAS316 AND SUGGESTIONS FOR FURTHER WORK

The major bottleneck in the use of the present system is the nonavailability of an efficient filing system. An editor to input Pascal programs and data is needed to make the system fully operational. A filing system and a Text Editor are currently under development in sister projects [1,3].

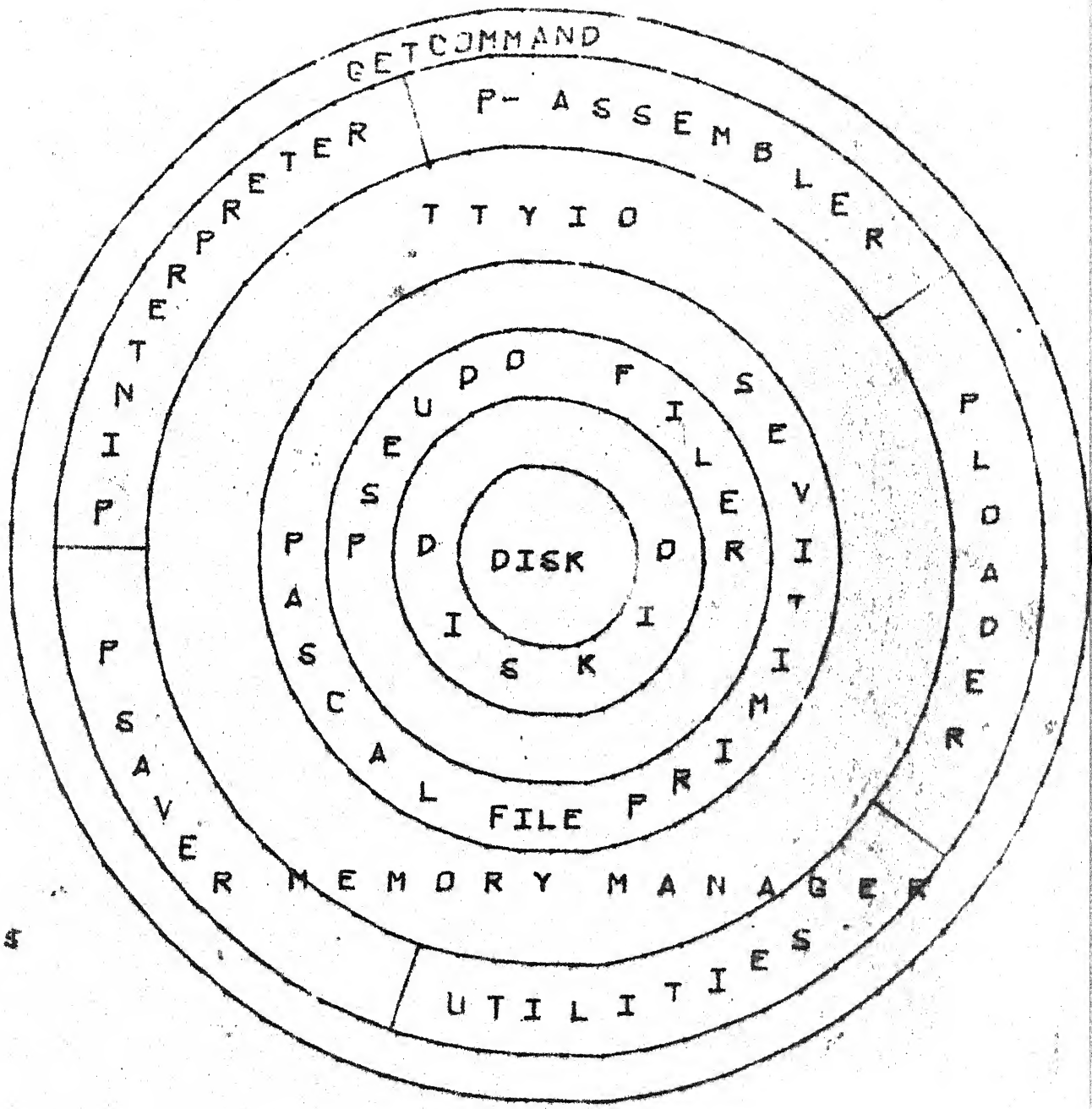


Fig. 5.1 Structure of PAS316 system

The present version of PAS316 does not support real arithmetic. A floating point processor or a floating point software package may be added to augment the system.

An integrated test of the complete P-system has not yet been made on TDC-316 because of lack of requisite utility systems. All the modules have been tested and found satisfactory on TDC-316. Also assembly and interpretation of Pascal programs have been successfully tested on a simulator of TDC-316 on DEC-10.

REFERENCES

1. Aarti Kumar, A filing system for TDC-316, Department of Computer Science, IIT-Kanpur, 1982, to be published.
2. Aho, A.V., and Ullman,J.D., Principles of Compiler Design, Addison-Wesley Publishing Company, 1979.
3. Ajay Tyagi, A Text Editor for TDC-316, Department of Computer Science, IIT Kanpur, 1982, to be published.
4. Berry,R.E., Experience with the Pascal P-compiler, Software - Practice and Experience, 8, No.5, 617-627, 1979.
5. Colemann, S.S., Pode, P.C., and Waite, W.M., The mobile programming system: JANUS, Software - Practice and Experience, 4, No.1, 5-23, 1974.
6. Digital Equipment Corporation, BLISS-11, Programmer's Manual, 1974.
7. Digital Equipment Corporation, PDP-11, Processor Handbook, 1971.
8. Eckhouse, R.H., Morris, R.L., Minicomputer Systems Organization, Programming and Applications (PDP-11), Prentice Hall Inc., Second Edition, 1979.
9. Electronics Corporation of India Limited, System Manual TDC-316, Volumes I,II and III.
10. Hurst, A.J., Pascal-P, Program structure and Program Behaviour, Software - Practice and Experience, 10, No.10, 1029-1036, 1980.

11. Jensen, K., and Wirth, N., Pascal - User Manual and Report, Springer-Verlag, Berlin, 1974.

12. Madnick, S.B, and Donovan, J.J., Operating Systems, McGraw-Hill Book Company, 1974.

13. Nori, K.V., Ammann, U., Jensen, K., Nageli, H.H., Jacobi, Ch., Pascal-P Implementation Notes, Nr.10, ETH Zurich.

14. Wirth, N., Algorithms + Data Structures = Programs, Prentice-Hall Inc., 1976.

15. Wirth, N., The Personal Computer, LILITH, Nr. 40, ETH Zurich.

APPENDIX - 1

The Language Pascal-P

The language processed by the Pascal-P compiler is 'standard Pascal' with some omissions and one change.

The features which are not processed are:

(a) procedures/functions as parameters,
(b) goto statements leading out of procedure/function bodies,
(c) all kind of files except predefined character files;
    The pre-defined standard text files are:
    INPUT,PRD (input files)
    OUTPUT,PRR (output files),
(d) all features associated with 'packing'.

Change:

(e) The standard procedure 'dispose' is replaced by 'mark' and 'release';
    MARK(P) where P is of any pointer type,marks the heap in the current state.
    RELEASE(P) releases all items created by a NEW instruction since the corresponding MARK(P).

APPENDIX - 2

!PASCAL-P COMPILER PERFORMANCE EVALUATION:

!TEST PROGRAM USED FOR THE FOLLOWING TESTS:

```
(*$l+,c+*)
(* binary tree traversal *)
PROGRAM traversal(input,output);
  TYPE
      ptr = ^node;
      node= RECORD
                  info :CHAR;
                  ll,rl : ptr
            END;
  VAR
      root :ptr; ch:CHAR;
      t:^INTEGER;

  PROCEDURE preorder(p:ptr);
        BEGIN
            IF p<> NIL THEN
              BEGIN write(output,p^.info);
                  preorder(p^.ll);
                  preorder(p^.rl)
              END
        END;
      (* preorder *)

  PROCEDURE inorder(p:ptr);
        BEGIN
            IF p <> NIL THEN
              BEGIN inorder(p^.ll);
                  write(output,p^.info);
                  inorder(p^.rl)
              END
        END;
      (* inorder *)

  PROCEDURE postorder(p:ptr);
        BEGIN
            IF p<> NIL THEN
              BEGIN postorder(p^.ll);
                  postorder(p^.rl);
                  write(p^.info)
              END
        END;
```

	RET	93	3607
*D	CSP	310	7000
	TXA	315	3180
	EQU	392	6704
	NEQ	197	1394
	GEQ	11	2728
	GRT	18	78
	LEQ	45	1977
	LES	17	1479
	UJP	800	7755
	FJP	788	20098
	XJP	23	749
	CHK	23	749
	EOF	1	1857
	ADT	92	3355
	ADR	0	0
	SBI	59	1391
	SBR	0	0
	SGS	2	0
	FLT	0	0
	FLO	0	0
	TRC	0	0
	NGI	4	0
	NGR	0	0
	SQI	0	0
	SQR	0	0
	ABI	0	0
	ABR	0	0
	NOT	99	4736
	AND	53	2047
	IOR	27	2163
	DIF	2	2
	INT	0	0
	UNI	114	411
	INN	81	1597
	MOD	1	183
	ODD	0	0
	MPI	8	10
	MPR	0	0
	DVI	2	2
	DVR	0	0
	MOV	212	335
	LCA	225	497
	DEC	229	2744
	STP	1	1
*N	LDL	1563	17652
*N	SIL	508	5555
*N	LAL	148	759
*N	UJC	37	0
	GET	0	0
	PUT	0	0
	RST	1	4
	RLN	0	0

NEW	58	81
WLN	57	291
WRS	68	533
FLN	1	1857
WRI	60	347
WRR	0	0
WRC	63	2026
RDI	0	0
RDR	0	0
RDC	1	1857
SIN	0	0
COS	0	0
EXP	0	0
LOG	0	0
SQT	0	0
ATN	0	0
SAV	1	4

The addition of new instructions and modification of existing one's has caused:

(a) Saving in CODE = 238 + 296 + 1557 + 505 + 148
= 2744 bytes.

(b) Saving in No. of Fetches =
2156 + 6952 + 17648 + 5553 + 759
= 33068

Procedure Call Statistics of the P-Compiler
****

Compilation of BINTRE.PAS

CALLING PROC.NAME	SIZE	CALLED PROC.NAME	SIZE	TIMES CALLED
**********	****	**********	****	**********
SYSTEM	3	MAIN	106	1
MAIN	106	INITSCAL	54	1
MAIN	106	INITSETS	24	1
MAIN	106	INITTABL	12	1
INITTABL	12	RESWORDS	266	1
INITTABL	12	SYMBOLS	314	1
INITTABL	12	RATORS	116	1
INITTABL	12	INSTRMNE	312	1
INITTABL	12	PROCMNEM	140	1
MAIN	106	ENTERSTD	104	1
MAIN	106	STDNAMES	212	1
MAIN	106	ENTSTDNA	495	1
ENTSTDNA	495	ENTERID	73	40
MAIN	106	ENTERUND	186	1
ENTERUND	186	GENLABEL	9	2
MAIN	106	INSYMBOL	401	1

INSYMBOL	401	NEXTCH	42	1066
NEXTCH	42	ENDOFLIN	130	67
INSYMBOL	401	OPTIONS	62	1
OPTIONS	62	NEXTCH	42	6
INSYMBOL	401	INIDENT	130	179
INIDENT	130	NEXTCH	42	785
MAIN	106	PROGRAMM	102	1
PROGRAMM	102	INSYMBOL	401	8
PROGRAMM	102	BLOCK	100	1
BLOCK	100	INSYMBOL	401	11
BLOCK	100	TYPEDECL	162	1
TYPEDECL	162	INSYMBOL	401	6
TYPEDECL	162	TYP	491	2
TYP	491	INSYMBOL	401	6
TYP	491	SEARCHID	100	2
TYPEDECL	162	ENTERID	73	2
TYP	491	FIELDLIS	490	1
FIELDLIS	490	ENTERID	73	3
FIELDLIS	490	INSYMBOL	401	7
FIELDLIS	490	TYP	491	2
TYP	491	SIMPLETY	350	4
SIMPLETY	350	SEARCHID	100	4
SIMPLETY	350	INSYMBOL	401	4
BLOCK	100	VARDECLA	189	1
VARDECLA	189	ENTERID	73	3
VARDECLA	189	INSYMBOL	401	9
VARDECLA	189	TYP	491	3
BLOCK	100	PROCDECL	408	4
PROCDECL	408	SEARCHSE	27	4
PROCDECL	408	GENLABEL	9	4
PROCDECL	408	ENTERID	73	4
PROCDECL	408	INSYMBOL	401	12
PROCDECL	408	PARAMETE	584	4
PARAMETE	584	INSYMBOL	401	21
PARAMETE	584	ENTERID	73	4
PARAMETE	584	SEARCHID	100	4
PROCDECL	408	BLOCK	100	4
BLOCK	100	BODY	417	5
BODY	417	PUTLABEL	14	5
BODY	417	GENLABEL	9	6
BODY	417	GENUJPEN	37	5
GENUJPEN	37	PUTIC	27	6
BODY	417	STATEMEN	194	35
STATEMEN	194	INSYMBOL	401	53
STATEMEN	194	IFSTATEM	54	4
IFSTATEM	54	EXPRESSI	261	4
EXPRESSI	261	SIMPLEEX	276	39
SIMPLEEX	276	TERM	205	39
TERM	205	FACTOR	391	39
FACTOR	391	SEARCHID	100	29
FACTOR	391	INSYMBOL	401	39
FACTOR	391	SELECTOR	450	25
EXPRESSI	261	LOAD	141	8
LOAD	141	GETADCH	62	34

LOAD	141	GEN1	152	34
GEN1	152	PUTIC	27	121
EXPRESSI	261	INSYMBOL	401	4
LOAD	141	GEN2	257	13
GEN2	257	PUTIC	27	32
EXPRESSI	261	COMPTYPE	228	4
COMPTYPE	228	COMPTYPE	228	4
EXPRESSI	261	GEN2	257	4
IFSTATEM	54	GENLABEL	9	5
IFSTATEM	54	GENFJP	50	4
GENFJP	50	LOAD	141	4
GENFJP	50	PUTIC	27	4
IFSTATEM	54	INSYMBOL	401	5
IFSTATEM	54	STATEMEN	194	5
STATEMEN	194	COMPOUND	33	4
COMPOUND	33	STATEMEN	194	13
STATEMEN	194	SEARCHID	100	45
STATEMEN	194	CALL	140	43
CALL	140	INSYMBOL	401	54
CALL	140	WRITE	307	23
WRITE	307	SEARCHID	100	23
WRITE	307	INSYMBOL	401	34
WRITE	307	EXPRESSI	261	13
SELECTOR	45	LOAD	141	12
SELECTOR	450	INSYMBOL	401	36
SELECTOR	450	SEARCHID	27	12
WRITE	307	LOAD	141	12
WRITE	307	GEN2	257	14
WRITE	307	GEN1	152	48
COMPOUND	33	INSYMBOL	401	13
CALL	140	CALLNONS	343	16
CALLNONS	343	GEN1	152	16
CALLNONS	343	INSYMBOL	401	32
CALLNONS	343	EXPRESSI	261	16
CALLNONS	343	LOAD	141	12
CALLNONS	343	COMPTYPE	228	28
CALLNONS	343	GENCUP	41	16
GENCUP	41	PUTIC	27	17
IFSTATEM	54	PUTLABEL	14	5
BODY	417	INSYMBOL	401	35
BODY	417	GEN1	152	6
CALL	140	READ	168	1
READ	168	SEARCHID	100	1
READ	168	INSYMBOL	401	2
READ	168	VARIABLE	22	1
VARIABLE	22	SEARCHID	100	4
VARIABLE	22	INSYMBOL	401	4
VARIABLE	22	SELECTOR	450	4
READ	168	LOADADDR	111	1
LOADADDR	111	GEN1	152	7
READ	168	GEN1	152	2
READ	168	COMPTYPE	228	3
CALL	140	NEW	152	1
NEW	152	VARIABLE	22	1

SELECTOR	450	GEN1	152	5
NEW	152	LOADADDR	111	1
NEW	152	GEN2	257	1
NEW	152	GEN1	152	1
STATEMEN	194	ASSIGNME	150	2
ASSIGNME	150	SELECTOR	450	2
ASSIGNME	150	LOADADDR	111	2
ASSIGNME	150	INSYMBOL	401	2
ASSIGNME	150	EXPRESSI	261	2
ASSIGNME	150	COMPTYPE	228	4
ASSIGNME	150	LOAD	141	2
ASSIGNME	150	STORE	75	2
STORE	75	GETADCH	62	2
STORE	75	GEN0	39	2
GEN0	39	PUTIC	27	3
CALLNONS	343	LOADADDR	111	4
IFSTATEM	54	GENUJPEN	37	1
CALL	140	MARK	26	1
MARK	26	VARIABLE	22	1
MARK	26	LOADADDR	111	1
MARK	26	GEN1	152	1
CALL	140	RELEASE	26	1
RELEASE	26	VARIABLE	22	1
RELEASE	26	LOAD	141	1
RELEASE	26	GEN1	152	1
WRITE	367	LOADADDR	111	1
LOADADDR	111	STRING	21	1
STRING	21	COMPTYPE	228	2
WRITE	367	STRING	21	1
BODY	417	GENCOD	41	1
BODY	417	GEN0	39	1

!Performance of the P-Compiler with segmented CODE Space
****

Compilation of BINTRE.PAS

Segment Size : 1024;
No. of virtual segments = 15;
No. of physical segments = 5;
Segment Replacement Policy : Passing the Parcel.

Compiler in its Original form :

S T A T I S T I C A L R E S U L T S
****

Total number of TRAPS: 1032

Number of Dummy TRAPS: 370

Number of Swap TRAPS: 662

Segment No.	Access Count	SwapinCount	SwapoutCount
0	123657	85	84
1	22[illegible]	82	82
2	1406	37	37
3	762	1	1
4	1145	5	5
5	[illegible]	[illegible]	107
[illegible]	[illegible]	62	61
7	3417	54	54
8	5461	51	51
9	2823	61	61
10	716	54	54
11	4375	58	57
12	2350	1	1
13	[illegible]	1	1
14	116	2	1

Compiler after Reorganization of procedures :

The following procedures have been grouped together:
endofline, nextch, inident, insymbol, enterid, searchid, putic, gen1, selector, write, call, factor, term, simpleexpression, expression, statement and body.

S I M U L A T I O N R E S U L T S
****

Total Number of TRAPS: 596

Number of Dummy TRAPS: 219

Number of Swap TRAPS: 377

Segment No.	Access Count	SwapinCount	SwapoutCount
0	140455	22	21
1	7781	36	36
2	2696	36	36
3	4094	56	55
4	13039	48	48
5	1730	39	39
6	640	2	2
7	1585	6	6
8	6129	43	42
9	1952	34	34
10	3003	38	38
11	675	13	12
12	2350	1	1

13	1806	1	1
14	116	2	1

APPENDIX - 3

!PAS316 System Errors

```
 1 : write attempted on a write protected disk
 2 : seek/search error ; non data transfer error
 3 : bus grant late
 4 : crcc error/read or write disable
 5 : synchronous error
 6 : drive unsafe
 7 : command check
10 : sector over run
11 : track over run
12 : non existant memory
13 : other errors
14 : attempt to get/read beyond EOF
15 : buffers exhausted cannot open file
16 : file has not been opened for write
17 : input data erroneous
20 : too many digits for an integer
```

!P-Interpreter Error Messages

```
0  : heap overflows stack
1  : illegal instruction
2  : not implemented
3  : error in case statement
4  : stack overflows heap
5  : value out of range
```

!P-Assembler Error Messages

```
  0 : code array overflow
  1 : label table overflow
  2 : duplicated label
  3 : illegal instruction
  4 : unimplemented instruction
```

```
!P-Compiler Error Messages
 ************************

  1:    error in simple type
  2:    identifier expected
  3:    'program' expected
  4:    ')' expected
  5:    ':' expected
  6:    illegal symbol
  7:    error in parameter list
  8:    'of' expected
  9:    '(' expected
 10:    error in type
 11:    '[' expected
 12:    ']' expected
 13:    'end' expected
 14:    ';' expected
 15:    integer expected
 16:    '=' expected
 17:    'begin' expected
 18:    error in declaration part
 19:    error in field-list
 20:    ',' expected
 21:    '*' expected


 50:    error in constant
 51:    ':=' expected
 52:    'then' expected
 53:    'until' expected
 54:    'do' expected
 55:    'to'/'downto' expected
 56:    'if' expected
 57:    'file' expected
 58:    error in factor
 59:    error in variable


101:    identifier declared twice
102:    low bound exceeds high bound
103:    identifier is not of appropriate class
104:    identifier not declared
105:    sign not allowed
106:    number expected
107:    incompatible subrange types
108:    file not allowed here
109:    type must not be real
110:    tagfield type must be scalar or subrange
111:    incompatible with tagfield type
112:    index type must not be real
113:    index type must be scalar or subrange
114:    base type must not be real
115:    base type must be scalar or subrange
116:    error in type of standard procedure parameter
```

```
117:    unsatisfied forward reference
118:    forward reference type identifier in variable declaration
119:    forward declared; repetition of parameter list not allowed
120:    function result type must be scalar, subrange or pointer
121:    file value parameter not allowed
122:    forward declared function; repetition of result type not allowed
123:    missing result type in function declaration
124:    F-format for real only
125:    error in type of standard function parameter
126:    number of parameters does not agree with declaration
127:    illegal parameter substitution
128:    result type of parameter function does not agree with declaration
129:    type conflict of operands
130:    expression is not of set type
131:    tests on equality allowed only
132:    strict inclusion not allowed
133:    file comparision not allowed
134:    illegal type of operand(s)
135:    type of operand must be boolean
136:    set element type must be scalar or subrange
137:    set element types not compatible
138:    type of variable is not array
139:    index type is not compatible with declaration
140:    type of variable is not record
141:    type of variable must be file or pointer
142:    illegal parameter substitution
143:    illegal type of loop control variable
144:    illegal type of expression
145:    type conflict
146:    assignment of files not allowed
147:    label type incompatible with selecting expression
148:    subrange bounds must be scalar
149:    index type must not be integer
150:    assignment to standard function is not allowed
151:    assignment to formal function is not allowed
152:    no such field in this record
153:    type error in read
154:    actual parameter must be a variable
155:    control variable must neither be formal nor nonlocal
156:    multidefined case label
157:    too many cases in case statement
158:    missing corresponding variant declaration
159:    real or string tag fields not allowed
160:    previous declaration was not forward
161:    again forward declared
162:    parameter size must be constant
163:    missing variant in declaration
164:    substitution of standard proc./func. not allowed
165:    multidefined label
166:    multideclared label
167:    undeclared label
168:    undefined label
169:    error in base set
170:    value parameter expected
```

APPENDIX - 4

```
! Syntax of the P-machine's Assembly Language in BNF.
  ***************************************************

<assembly program> ::= <assembly record> <assembly record>
                       @ see note i
<assembly record> ::= <assembly segment> END_OF_LINE
<assembly segment> ::= <assembly statement> END_OF_LINE
                       <assembly segment> | LAMBDA
<assembly statement> ::= I<integer> PPC<integer>|
                         @ see note ii
                         L<integer>| L<integer> "=" <integer>|
                         " "<assembly instruction>.
<relational instruction> ::= <opcode class 5> <char 1>|
                             <opcode class 5> M<integer>
<procedure call instruction> ::= CSP<standard procedure mnemonic>|
                                 CUP <integer> L<integer>
<return instruction> ::= RET<return what>
<return what> ::= P|R|I|B|C|A
<load constants instruction> ::= <load string> | <load others>
<load string> ::= LCA<string>
<load others> ::= LDC<load what>
<load what> ::= I<integer>| R<real>| B<integer>|N|
                "("<integer list>")"
<integer list> ::= <integer> <integer list> | LAMBDA
<ent instruction> ::= ENT<integer> L<integer>
<assembly instruction> ::= <opcode class 1>|
                           <opcode class 2><integer>|
                           <opcode class 3><integer><integer>|
                           <opcode c Lass 4> L<integer>|
                           <opcode class 6><char 1>|
                           <opcode class 7><char 1><integer>|
                           <opcode class 8><char 1><integer><integer>|
                           <relational instruction>|
                           <procedure call instruction>|
                           <return instruction>|
                           <load constants instruction>|
                           <ent instruction>
<opcode class 1> ::= ABI|ABR|ADI|ADR|AND|DEC|DIF|DVI|DVR|EOF|
                     FLT|FLO|INN|IOR|INC|INT|MOD|MPI|MPR|NGI|
                     NOT|ODD|SBI|SBR|SGS|SQI|SQR|UJC|STP|TRC|UNI
<opcode class 2> ::= IXA|LAO|LAL|MOV|MST
<opcode class 3> ::= LDA
<opcode class 4> ::= UJP|XJP|FJP
<opcode class 5> ::= EQU|NEQ|GEQ|LEQ|GRT|LES
<opcode class 6> ::= STO
```

```
<opcode class 7> ::= LDL|LDO|STL|SRO|IND
<opcode class 8> ::= LOD|STR|CHK
<char 1>    ::= A|I|R|B|S
<standard procedure mnemonic> ::= GET|PUT|ELN|NEW|WRS|WLN|WRI|
                                  WRR|WRC|WRO|RDI|RDR|RDC|RST|
                                  SAV|SIN|COS|EXP|LOG|SQT|ATN|RLN
```

<integer>,<string>,<real> and END_OF_LINE are as defined in Pascal. The terminal symbols are either enclosed in quotes (Ex: "=") or written in Upper case (Ex: I).

Note:

(i) The first assembly record of the assembly program consists of the whole Pascal program and should be loaded at PCODE[7] of the P-Machine. The second record, to be loaded at PCODE[0], consists of a Call to the outermost block of the Pascal program.

(ii) I<integer> indicates the number of the instruction to be generated. PPC<integer> indicates the value of the P-Machine's Program Counter at which the next instruction should be assembled. This statement is generated for every tenth instruction by the Compiler, to allow the reader of the code to relate it with the source program listing produced by the compiler.

APPENDIX - 5

P-Machine's Instruction Set.

CODE	MNEMONIC	PARAMETERS	DESCRIPTION
0	EOLN		Checks the EOLN condition for the file specified on the top of the stack;the result of this check is left on the top of the stack.
1	GET		Performs GET on the file specified by the top of the stack and appropriately fills the buffer associated with it.
2	RDC		Reads a character from the file specified on the top of the stack and assigns it to the variable whose address is below the top of the stack;note the automatic updating of the buffer associated with the file.
3	RDI		Reads an integer from the file specified on the top of the stack and assigns it to a variable whose address is below the top of the stack;note the automatic updating of the buffer associated with the file.
4	RDR		Reads a real number from the file specified on the top of the stack and assigns it to a variable whose address is below the top of the stack;note the automatic updating of the buffer associated with the file.
5	RLN		The top of the stack specifies a file on which a READLN is performed; note the automatic updating of the buffer associated with the file.
6	PUT		Performs the PUT operation on the file specified by the top of stack.
7	WRC		Writes on the file specified on the top of the stack a character which is found immediately below the element below the top of the stack. Just below the top is the number of characters to be written out.~
8	WRI		Writes on the file specified by the top of the stack an integer whose value is given immediately below the element below the top of the stack. Just below the top is the number of characters to be written out.~
9	WRO		Same as WRI, but Octal digits are written out.~
10	WRR		Writes on the file specified by the top of the stack a real number whose value is given immediately below the element below the top of the stack. Just below the top is the number of characters to be written out.~
11	WRS		Writes on the file specified by the top of the stack a string of characters; immediately below the top of the stack is the value of the actual length of the string;below this is specified the actual length to be written out.~ Before WRS is executed the string to be written must be loaded into the P-Machine's array STRING.

12	WLN	Performs WRITELN on the file specified on the top of the stack.
13	NEW	The top of the stack specifies the size of element to be allocated from the free stroage;the address of the element is to be stored in the pointer variable whose address is to be found below the top of the stack.
14	PAK	?
15	RST	Sets the 'NEW' pointer (heap pointer) to the pointer value on the top of the stack.
16	SAV	Saves the current value of the 'NEW' pointer (heap pointer) at the address specified on the top of the stack.
17	SIN	Top of the stack := SINE(top of the stack).
18	COS	Top of the stack := COSINE(top of the stack).
19	ATN	Top of the stack := ARCTANGENT(top of the stack).
20	EXP	Top of the stack := e ^ top of the stack.
21	LOG	Top of the stack := Natural Logarithm of top of the stack.
22	SQT	Top of the stack := Square Root of top of the stack.
32	EOF	test on end of file
33	NOT	Boolean NOT
34	IOR	Boolean 'inclusive or'
35	AND	Boolean AND
36	LDCBF	Load constant Boolean FALSE
37	LDCBT	Load constant Boolean TRUE
38	LESB	compare Boolean less than
39	LEQB	compare Boolean less than or equal
40	EQUB	compare Boolean equal
41	NEQB	compare Boolean not equal
42	GEQB	compare Boolean greater than or equal
43	GRTB	compare Boolean greater than
44	ODD	test on odd
45	NGI	integer sign inversion
46	ADI	integer addition
47	SBI	integer subtraction
48	MPI	integer multiplication
49	DVI	integer division
50	MOD	modulus
51	SQI	square integer
52	ABI	absolute value of integer
53	LDCN	load constant NIL
54	LESA	compare address less than
54	LESI	compare integer less than
55	LEQA	compare address less than or equal
55	LEQI	compare integer less than or equal
56	EQUA	compare address equal
56	EQUI	compare integer equal
57	NEQA	compare address not equal
57	NEQI	compare integer not equal
58	GEQA	compare address greater than or equal
58	GEQI	compare integer greater than or equal
59	GRTA	compare address greater than
59	GRTI	compare integer greater than
60	SGS	generate singleton set
61	INN	test set membership (IN)

62	UNI			set union
63	INT			set intersection
64	DIF			set difference
65	LESS			compare Set less than
66	LEQS			compare Set less than or equal
67	EQUS			compare Set equal
68	NEQS			compare Set not equal
69	GEQS			compare Set greater than or equal
70	GRTS			compare Set greater than
71	TRC			truncation
72	FLO			float next to the top
73	FLT			float top of the stack
74	NGR			real sign inversion
75	ADR			real addition
76	SBR			real subtraction
77	MPR			real multiplication
78	DVR			real division
79	SQR			square real
80	ABR			absolute value of real number
81	LESR			compare Real less than
82	LEQR			compare Real less than or equal
83	EQUR			compare Real equal
84	NEQR			compare Real not equal
85	GEQR			compare Real greater than or equal
86	GRTR			compare Real greater than
87	UJC			error in case statement
88	STP			stop
133	LESM		Q	compare multiple units less than
134	LEQM		Q	compare multiple units less than or equal
135	EQUM		Q	compare multiple units equal
136	NEQM		Q	compare multiple units not equal
137	GEQM		Q	compare multiple units greater than or equal
138	GRTM		Q	compare multiple units greater than
139	DEC		Q	decrement
140	INC		Q	increment
141	IXA		Q	compute indexed address
142	LAL		Q	load current-level address
143	LAO		Q	load base-level address
144	FJP		Q	false jump
145	XJP		Q	indexed jump
146	UJP		Q	unconditional jump
147	ENT		Q	enter block
148	MOV		Q	move
149	LDCI		Q	load constant integer
150	LDA	P	Q	load address with level P
151	CUP	P	Q	call user procedure
152	MST	P		mark stack
153	RET	P		return from block
154	CHK	L	Q	check against upper and lower bounds

155	LCA	STRING	0	load string,the string is terminated by a 0
192	LDCR	R		load constant real
193	LDCS	S		load constant set
194	LNDA		0	indexed fetch address
194	LNDI		0	indexed fetch integer
194	LNDB		0	indexed fetch boolean
195	LNDR		0	indexed fetch real
196	LNDS		0	indexed fetch set
197	LDLA		0	load contents of current-level address address
197	LDLI		0	load contents of current-level address integer
197	LDLB		0	load contents of current-level address boolean
198	LDLR		0	load contents of current-level address real
199	LDLS		0	load contents of current-level address set
200	LDDA		0	load contents of base-level address address
200	LDDI		0	load contents of base-level address integer
200	LDDB		0	load contents of base-level address boolean
201	LDDR		0	load contents of base-level address real
202	LDDS		0	load contents of base-level address set
203	LODA	P	0	load contents of address address
203	LODI	P	0	load contents of address integer
203	LODB	P	0	load contents of address boolean
204	LODR	P	0	load contents of address real
205	LODS	P	0	load contents of address set
206	SIDA			store indirect address
206	SIDI			store indirect integer
206	SIDB			store indirect boolean
207	SIDR			store indirect real
208	SIDS			store indirect set
209	SILA		0	store at current-level address address
209	SILI		0	store at current-level address integer
209	SILB		0	store at current-level address boolean
210	SILR		0	store at current-level address real
211	SILS		0	store at current-level address set
212	SRDA		0	store at base-level address address
212	SRDI		0	store at base-level address integer
212	SRDB		0	store at base-level address boolean
213	SRDR		0	store at base-level address real
214	SRDS		0	store at base-level address set
215	SIRA	P	0	store at address address
215	SIRI	P	0	store at address integer
215	SIRB	P	0	store at address boolean
216	SIRR	P	0	store at address real
217	SIRS	P	0	store at address set

Note:
(i) : if necessary leading blanks are filled in.
(ii) Parameter Section
P : one byte
O : two bytes
L : two bytes
R : four bytes(real)
S : eight bytes(set)

APPENDIX - 6

PAS316

A SINGLE USER PASCAL PROGRAMMING STATION

```
00010  %
00020  ************************************************************************
00030  **********                                                    **********
00040  **********  MODULE : PAS316                                   **********
00050  **********  DATE : 1 JULY 1982                                **********
00060  **********  AUTHOR : S SRINIVAS RAGHURAM                      **********
00070  **********  MAIN MODULE : Command Processor                   **********
00080  **********  VALID COMMANDS :                                  **********
00090  **********   ASMINT : Assemble and Interpret                  **********
00100  **********   COPY   : Copy ASCII files                        **********
00110  **********   PASASM : Compile and Assemble                    **********
00120  **********   PASCAL : Compile                                 **********
00130  **********   PASGO  : Compile, Assemble and Interpret         **********
00140  **********   ASM    : Assemble                                **********
00150  **********   EX     : Interpret                               **********
00160  **********   PRINT  : Print file on line printer              **********
00170  **********   TYPE   : Type file on tty                        **********
00180  **********   KJOB   : Exit from PAS316                        **********
00190  **********                                                    **********
00200  ************************************************************************%
00210  MODULE pas316( MAIN , STACK , LIST )=
00220   BEGIN
00230
00240     MACRO lsby = 0,8$;
00250
00260     BIND cmptrk = 33, cmpsrfc = 0, cmpsectr = 1;
00270           ! P-Compiler's P-CODE disk file address
00280     BIND false = 0, true = 1,
00290          space = #40, slash = #57, aye = #101, yell = #114, owe = #117,
00300          yes = #123, tee = #124, zed = #132, save = #141,
00310          szed = #172, cr = #15, lf = #12, ctrlu = #25, star = #52,
00320          stkptr = #177742, cmdtabsize = 10;
00330     BIND version = UPLIT ASCIZ "PAS316 Pascal System Version of Jul 1,82",
00340          compver = UPLIT ASCIZ "PAS316 P-Compiler Version of Jul 1,82",
00350          mes1    = UPLIT ASCIZ "Give Object",
00360          mes2    = UPLIT ASCIZ "Illegal Command",
00370          mes3    = UPLIT ASCIZ "Give Source",
00380          mes4    = UPLIT ASCIZ "Give Destn";
00390
00400     STRUCTURE cmmdtable[ i ] =
00410      (.cmmdtable + .i * 6);
00420
00430     BIND cmmdtable cmdtab = UPLIT(
00440      "      " , "ASMINT" , "COPY  " , "PASASM" , "PASCAL" , "PASGO " ,
00450            "ASM   " , "EX    " , "PRINT " , "TYPE  " , "KJOB  " );
00460
00470
00480     GLOBAL savstkval;
00490     GLOBAL ttyno;
00500
00510     BYTE OWN VECTOR ID[ 6 ];
00520     OWN ch , k , t , c , s;
00530     OWN list , trace , obj , ttyout , abrt , notquit;
00540
```

```
00550         EXTERNAL getlcn,ttcrlf,ttoutstr,putlch;
00560         EXTERNAL ttinit,diskinit,inittraps,initmap,initbufs;
00570         EXTERNAL asm,intr,psave,pload;
00580         EXTERNAL getfilename,copy,type;
00590
00600         ROUTINE getch =
00610          BEGIN
00620           getlch(.ttyno,ch);
00630           IF .ch GEQ save THEN
00640            IF .ch LEQ szed THEN ch = .ch - #40;
00650            IF .ch EQL cr THEN putlch(.ttyno,lf)
00660             ELSE IF .ch EQL lf THEN putlch(.ttyno,cr)
00670              ELSE IF .ch EQL ctrlu THEN abrt = true;
00680          END; ! getch
00690
00700         ROUTINE negcomp =
00710          BEGIN
00720           LOCAL c,b;
00730           c = 0; b = true;
00740           WHILE .b AND (.c LEQ 5) DO
00750            IF .id[.c]<lsby> EQL .(cmdtab[.kl+.c)<lsby> THEN c = .c + 1
00760              ELSE b = false;
00770           RETURN not .b
00780          END; ! negcomp
00790
00800         ROUTINE setswitches =
00810          BEGIN
00820           DO
00830            BEGIN
00840             getch();
00850             IF .ch EQL yell THEN list = NOT .list
00860              ELSE IF .ch EQL tee THEN trace = NOT .trace
00870               ELSE IF .ch EQL yes THEN obj = NOT .obj
00880                ELSE IF .ch EQL owe THEN ttyout = NOT .ttyout;
00890             getch()
00900            END
00910           UNTIL .ch NEQ slash
00920          END; ! setswitches
00930
00940         ROUTINE compile =
00950          BEGIN
00960           pload(cmptrk,cmpsrfc,cmpsectr);
00970           ttcrlf(.ttyno); ttoutstr(.ttyno,compver);
00980           intr(.trace)
00990          END; ! compile
01000
01010         ROUTINE assemble =
01020          BEGIN
01030           LOCAL n;
01040           n = asm(.list);
01050           IF .obj THEN
01060            BEGIN
01070             ttcrlf(.ttyno); ttoutstr(.ttyno,mes1);
01080             getfilename(t,c,s);
```

```
01090            psave(.t,.c,.s,.n)
01100           END
01110         END; ! assemble
01120
01130        GLOBAL ROUTINE getcommand =
01140         BEGIN
01150          ttcrlf(.ttyno);
01160          putlch(.ttyno,star); ch = space;
01170          list = trace = obj = abrt = false; ttyout = true;
01180          INCR i FROM 0 TO 5 DO idf[.i] = .ch;
01190          WHILE .ch EQL space DO getch();
01200          IF .ch GEQ aye AND .ch LEQ zed THEN
01210           BEGIN
01220            LOCAL ind;
01230            ind = 0;
01240            WHILE .ch GEQ aye AND .ch LEQ zed DO
01250             BEGIN
01260              IF .ind LEQ 5 THEN idf[.ind] = .ch;
01270              ind = .ind + 1;
01280              getch()
01290             END;
01300            INCR i FROM 0 TO 5 DO
01310             (cmdtab[0] + .i)<lsby> = .idf[.i];
01320            k = cmdtabsize;
01330            WHILE negcomp() DO k = .k - 1;
01340            IF .ch EQL slash THEN setswitches();
01350            IF .abrt THEN RETURN;
01360            CASE .k OF
01370             SET
01380              !0
01390              BEGIN
01400               ttcrlf(.ttyno); ttoutstr(.ttyno,mes2)
01410              END;
01420
01430              !1 ASMINT
01440              BEGIN
01450               assemble(); intr(.trace,.ttyout)
01460              END;
01470
01480              !2 COPY
01490              BEGIN
01500               LOCAL t1,c1,s1;
01510               ttcrlf(.ttyno); ttoutstr(.ttyno,mes3);
01520               getfilename(t,c,s);
01530               ttcrlf(.ttyno); ttoutstr(.ttyno,mes4);
01540               getfilename(t1,c1,s1);
01550               copy(.t,.c,.s,.t1,.c1,.s1)
01560              END;
01570
01580              !3 PASASM
01590              BEGIN
01600               compile(); obj = NOT .obj; assemble()
01610              END;
01620
```

```
01630             !4 PASCAL
01640              compile();
01650
01660             !5 PASGO
01670             BEGIN
01680              compile(); assemble(); intr(.trace,.ttyout)
01690             END;
01700
01710             !6 ASM
01720             BEGIN
01730              obj = NOT .obj; assemble()
01740             END;
01750
01760             !7 EX
01770             BEGIN
01780              ttcrlf(.ttyno); ttoutstr(.ttyno,mes1);
01790              getfilename(t,c,s);
01800              pload(.t,.c,.s);
01810              intr(.trace,.ttyout)
01820             END;
01830
01840             !8 PRINT
01850             ;
01860
01870             !9 TYPE
01880             BEGIN
01890              ttcrlf(.ttyno); getfilename(t,c,s); type(.t,.c,.s)
01900             END;
01910
01920             !10 KJOB
01930              notquit = false
01940
01950           TES;
01960         END ! IF
01970       END; ! getcommand
01980
01990       ! MAIN program of PAS316
02000        reset();
02010        ttyno = 4;
02020        initmap();
02030        inittraps();
02040        ttinit();
02050        diskinit();
02060        initbufs();
02070        ttcrlf(.ttyno); ttoutstr(.ttyno,version);
02080        notquit = true;
02090        savstkval = .stkptr - 2;
02100        WHILE .notquit DO getcommand() ;
02110     END
02120   ELUDOM
```

```
00010       %
00020       ************************************************************
00030       **********                                                  **********
00040       **********  MODULE : PASSEMBLER                             **********
00050       **********  DATE : 1 JULY 1982                              **********
00060       **********  AUTHOR : S SRINIVAS RAGHURAM                    **********
00070       **********  INPUT  : P-Machine's Assembly Language Program  **********
00080       **********             in file PRR.                         **********
00090       **********  OUTPUT : The object code is loaded into the     **********
00100       **********             P-Machine's code array PCODE.        **********
00110       **********  LSTFIL : Listing is optionally produced.        **********
00120       **********  ENTRY  : ASM(LSTN);                             **********
00130       **********             lstn = true then assembly listing    **********
00140       **********             is generated.                        **********
00150       **********  RETURNS: Maximum value of Location Counter PPC  **********
00160       **********                                                  **********
00170       ************************************************************%
00180       MODULE passembler( unsafe , nooptimize ,nofinal ,   nolist   ) =
00190
00200        BEGIN
00210         % The P-Assembler has almost no error detection or error recovery.
00220           It expects a clean and correct input , typically produced by a
00230           Compiler.                                                    %
00240
00250         %              Assembler ERRORS:
00260                           0 : CODE ARRAY OVERFLOW
00270                           1 : LABEL TABLE OVERFLOW
00280                           2 : DUPLICATED LABEL
00290                           3 : ILLEGAL INSTRUCTION
00300                           4 : UNIMPLEMENTED INSTRUCTION
00310                                                                    %
00320
00330        MACRO
00340         lsby = 0,8 $,      ! least significant byte
00350         msby = 8,8 $;      ! most significant byte
00360
00370        BIND
00380         ! ASCII characters
00390         aye  = #101, bee  = #102, sea  = #103, eye  = #111, ell = #114,
00400         emm  = #115, yen  = #116, pee  = #120, are  = #122, ess  = #123,
00410         lpar = #50 , rpar = #51 , quote= #47 , space= #40 , nul  = #0  ,
00420         zero = #60 , nine = #71 , plus = #53 , minus= #55 , equal= #75 ,
00430
00440         ! P-Machine OpCodes
00450         LDCIP  = 149, LDCRP = 192, LDCNP = 53, LDCBFP= 36, LDCBTP= 37,
00460         LDCSP  = 193, CSPP  = 0  , CHKP  =154, LDCP  = 36, CUPP  =151,
00470         RETP   = 153, LCAP  = 155, INDP  =194, LDLP  =197, LDOP  =200,
00480         STLP   = 209, SROP  = 212, LODP  =203, STRP  =215, LESP  = 38,
00490         LEQP   =  39, EQUP  =  40, NEQP  = 41, GEQP  = 42, GRTP  = 43,
00500         DECP   = 139, INCP  = 140, IXAP  =141, LALP  =142, LAOP  =143,
00510         MOVP   = 148, FJPP  = 144, XJPP  =145, UJPP  =146, ENTP  =147,
00520         STOP   = 206, LDAP  = 150, MSTP  =152,
00530
00540         ! Constants
```

```
00550         w1 = 0, w2 = 1, false = 0, true  = 1, entered = 0, defined = 1,
00560         nil=-1,nois=61, ! number of instructions - 1
00570               nosfp=22, ! number of special functions/procedures - 1
00580         codemax = #40000, ! size of P-Machine's CODE SEGMENT
00590         maxlabel=   1999, ! maximum number of labels assembler can handle
00600         begincode = 7, ! start of code of segment1, segment2 starts at 0
00610         prr = 3,  ! input file for assembler
00620         lstfil = 2; ! listfile produced by assembler
00630
00640       BIND
00650         version = UPLIT ASCIZ "PAS316 P-Assembler Version of Jul 1,82",
00660         mes1    = UPLIT ASCIZ "End of Assembly",
00670         err0    = UPLIT ASCIZ "CODE ARRAY OVERFLOW",
00680         err1    = UPLIT ASCIZ "LABEL TABLE OVERFLOW",
00690         err2    = UPLIT ASCIZ "DUPLICATED LABEL",
00700         err3    = UPLIT ASCIZ "ILLEGAL INSTRUCTION",
00710         err4    = UPLIT ASCIZ "UNIMPLEMENTED INSTRUCTION";
00720
00730       % to understand the expansion of the following 2 MACRO's refer
00740         Bliss-11 Programmer's Manual pp52                          %
00750       MACRO
00760         initopcodes( aryname )[] =
00770           loadopcode( aryname , $REMAINING ) $;
00780       MACRO
00790         loadopcode( start )[ opval ] =
00800           start[ $COUNT ] = opval $;
00810
00820       STRUCTURE insttable[ i , field ]=
00830         ( .insttable + .i * 4 + .field * 2 );
00840
00850
00860       BIND insttable inst = UPLIT(
00870         "ABI ", "ABR ", "ADI ", "ADR ", "AND ", "CHK ", "CSP ", "CUP ",
00880         "DEC ", "DIF ", "DVI ", "DVR ", "ENT ", "EOF ", "EQU ", "FJP ",
00890         "FLO ", "FLT ", "GEQ ", "GRT ", "INC ", "IND ", "INN ", "INT ",
00900         "IOR ", "IXA ", "LAL ", "LAO ", "LCA ", "LDA ", "LDC ", "LDL ",
00910         "LDO ", "LEQ ", "LES ", "LOD ", "MOD ", "MOV ", "MPI ", "MPR ",
00920         "MST ", "NEQ ", "NGI ", "NGR ", "NOT ", "ODD ", "RET ", "SBI ",
00930         "SBR ", "SGS ", "SQI ", "SOR ", "SRO ", "STL ", "STO ", "STP ",
00940         "STR ", "TRC ", "UJC ", "UJP ", "UNI ", "XJP "                  );
00950
00960       BIND insttable sptab = UPLIT(
00970         "ELN ", "GET ", "RDC ", "RDI ", "RDR ", "RLN ", "PUT ", "WRC ",
00980         "WRI ", "WRO ", "WRR ", "WRS ", "WLN ", "NEW ", "PAK ", "RST ",
00990         "SAV ", "SIN ", "COS ", "ATN ", "EXP ", "LOG ", "SQT "          );
01000
01010       EXTERNAL endoffile , endofline, fileclose;
01020       EXTERNAL ttoutstr , ttputoct , ttcrlf , putlch;
01030       EXTERNAL read , write , readln , writeln , readint , reset , rewrite;
01040       EXTERNAL getfilename , deposit , fetch;
01050       EXTERNAL ttyno;
01060
01070
01080       OWN ppc ,    ! P-Machine's Program Counter
```

```
01090            op ,     ! OPcode
01100             p ,     ! P register
01110             q ,     ! Q register
01120             k ,     ! index variable used for searching tables
01130            cn ,     ! look ahead character for file PRR
01140      labelvalue;     ! label value for updating labeltable
01150
01160      OWN    prradr;  ! buffer for file PRR
01170
01180      OWN    lstadr;  ! buffer for file LSTFIL
01190
01200      OWN listing , prnt;
01210
01220      BYTE OWN VECTOR opc[ nois ];
01230
01240      BYTE OWN VECTOR lblst[ maxlabel + 1 ];
01250
01260      OWN VECTOR lblval[ maxlabel + 1 ];
01270
01280      OWN VECTOR id[ 2 ];
01290
01300       ROUTINE initassembler =
01310       BEGIN
01320        LOCAL trk,srfc,sectr;
01330        initopcodes( OPC ,
01340         52 ,80 ,46 ,75 ,35 ,154,0  ,151,139,64 ,49 ,78 ,147,32 ,40 ,
01350         144,72 ,73 ,42 ,43 ,140,194,61 ,63 ,34 ,141,142,143,155,150,
01360         36 ,197,200,39 ,38 ,203,50 ,148,48 ,77 ,152,41 ,45 ,74 ,33 ,
01370         44 ,153,47 ,76 ,60 ,51 ,79 ,212,209,206,88 ,215,71 ,87 ,146,
01380         62 ,145                                                    );
01390
01400        ppc = begincode;
01410        prnt = true;
01420        ttcrlf(.ttyno); getfilename(trk,srfc,sectr);
01430        reset(prr,prradr,.trk,.srfc,.sectr);
01440        IF .listing THEN
01450         BEGIN
01460          ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Listing");
01470          getfilename(trk,srfc,sectr);
01480          rewrite(lstfil,lstadr,.trk,.srfc,.sectr);
01490         END;
01500        id[ w1 ]<lsby> = id[ w1 ]<msby> = id[ w2 ]<lsby> = id[ w2 ]<msby> = space;
01510        INCR i FROM 0 TO maxlabel DO
01520         BEGIN
01530          lblval[ .i ] = nil; lblst[ .i ] = entered;
01540         END;
01550       END; ! initassembler
01560
01570        ROUTINE asserr( n )=
01580        BEGIN
01590         LOCAL errmesg;
01600         ttcrlf(.ttyno);
01610         ttoutstr(.ttyno,UPLIT ASCIZ "Assembly Error: ");
01620         ttputoct(.ttyno,.n);
```

```
01630           ttcrlf(.ttyno);
01640           CASE .n OF
01650            SET
01660              errmesg = err0;
01670              errmesg = err1;
01680              errmesg = err2;
01690              errmesg = err3;
01700              errmesg = err4
01710            TES;
01720           ttoutstr(.ttyno,.errmesg); ttcrlf(.ttyno);
01730           EMT(0);
01740         END; ! asserr
01750
01760        MACRO octch(f,dig) = write(f,zero+dig)$;
01770
01780        ROUTINE writbyte(f,value)=
01790         BEGIN
01800          LOCAL dig;
01810          dig = .value<6,2>; octch(.f,.dig);
01820          dig = .value<3,3>; octch(.f,.dig);
01830          dig = .value<0,3>; octch(.f,.dig)
01840         END; ! writbyte
01850
01860        ROUTINE writword(f,value)=
01870         BEGIN
01880          LOCAL dig;
01890          dig = .value<15,1>; octch(.f,.dig);
01900          INCR i FROM 1 TO 5 DO
01910           BEGIN
01920            dig = .value<12,3>; octch(.f,.dig);
01930            value = .value * 8
01940           END
01950         END; ! writword
01960
01970         ROUTINE putword( adr , value )=
01980         BEGIN
01990          LOCAL tmp;
02000          tmp = .value<lsby>; deposit(.adr,.tmp);
02010          tmp = .value<msby>; deposit(.adr+1,.tmp)
02020         END; !putword
02030
02040         ROUTINE getword( adr )=
02050         BEGIN
02060          LOCAL value;
02070           value<lsby> = fetch( .adr );
02080           value<msby> = fetch( .adr + 1 );
02090           RETURN .value;
02100         END; ! getword
02110
02120         ROUTINE loadbyte( value )=
02130         BEGIN
02140          IF .listing THEN
02150           IF .prnt THEN
02160            BEGIN
```

```
02170            INCR i FROM 0 TO 3 DO
02180             write(lstfil,space);
02190             writbyte(lstfil,.value)
02200      !%debug% ; putlch(.ttyno,space);  ttputoct(.ttyno,.value); putlch(.ttyno,space);
02210           END;
02220        deposit(.dpc,.value);
02230        dpc = .dpc + 1;
02240        IF .dpc GTR codemax THEN asserr(0);
02250       END; ! loadbyte
02260
02270       ROUTINE loadword( value )=
02280       BEGIN
02290        LOCAL val;
02300        IF .listing THEN
02310         BEGIN
02320          write(lstfil,space);
02330          writword(lstfil,.value);
02340      !    %debug% putlch(.ttyno,space); ttputoct(.ttyno,.value);
02350          prnt = false
02360         END;
02370        val = .value<lsby>;
02380        loadbyte( .val );
02390        val = .value<msby>;
02400        loadbyte( .val );
02410        IF .listing THEN  prnt=true;
02420       END; ! loadword
02430
02440
02450       ROUTINE update( x ) =
02460        ! when a label definition LX is found
02470       BEGIN
02480        LOCAL curr , succ , endlist;
02490        IF .x GTR maxlabel THEN asserr(1);
02500        IF .lblst[ .x ] EQL defined THEN asserr( 2 )
02510         ELSE
02520          BEGIN
02530           IF .lblval[ .x ] NEQ nil THEN
02540            BEGIN
02550             curr = .lblval[ .x ]; endlist = false;
02560             UNTIL .endlist DO
02570              BEGIN
02580               succ = getword( .curr );
02590               putword( .curr , .labievalue );
02600               IF .succ EQL nil THEN endlist = true
02610                ELSE curr = .succ;
02620              END
02630            END;
02640           lblst[ .x ] = defined;
02650           lblval[ .x ] = .labievalue;
02660          END
02670         END; ! update
02680
02690       ROUTINE lookup( x ) =
02700       ! search in label table
```

```
02710          BEGIN
02720           IF .x GTR maxlabel THEN asserr(1);
02730           IF .lblst[ .x ] EQL entered THEN
02740            IF .lblval[ .x ] EQL nil THEN
02750             BEGIN
02760              lblval[ .x ] = .ppc;
02770              loadword(nil);
02780             END
02790            ELSE
02800             BEGIN
02810              LOCAL temp;
02820              temp = .ppc;
02830              loadword( .lblval[ .x ] );
02840              lblval[ .x ] = .temp;
02850             END
02860           ELSE  ! .lblst[ .x ] EQL defined
02870            loadword( .lblval[ .x ] );
02880          END; ! lookup
02890
02900          ROUTINE labelsearch =
02910          BEGIN
02920           LOCAL x;
02930           WHILE .ch NEQ yell AND NOT endofline(prr) DO
02940            read( prr , ch );
02950           readint(prr , x);
02960           lookup( .x );
02970          END; ! labelsearch
02980
02990        FORWARD assembler;
03000
03010          ROUTINE generate =
03020          ! generate segment of code
03030          BEGIN
03040           LOCAL x;  ! label number
03050           UNTIL endofline( prr ) DO
03060            BEGIN
03070             read( prr , ch );
03080             IF .ch EQL eye THEN readln( prr )
03090              ELSE
03100               IF .ch EQL yell THEN
03110                BEGIN
03120                 readint( prr , x );
03130                 IF NOT endofline( prr ) THEN read( prr , ch );
03140                 IF .ch EQL equal THEN readint( prr , lablevalue )
03150                  ELSE lablevalue = .ppc;
03160                 update( .x ); readln( prr );
03170                END
03180                ELSE ! .ch EQL space
03190                 BEGIN
03200                  DO read( prr , ch ) UNTIL .ch NEQ space;
03210                  IF .listing THEN
03220                   BEGIN
03230                    writeln(lstfil); writword(lstfil,.ppc);
03240 !                  %debug% ttcrlf(.ttyno); ttputoct(.ttyno,.ppc); putch(.ttyno,space);
```

```
03250                 write(lstfil,quote)
03260                END;
03270               assemoler();
03280              END
03290        END  ! until
03300     END; ! generate
03310
03320     ROUTINE assembler =
03330     BEGIN
03340
03350       ROUTINE getid =
03360       ! check <msby> & <lsby> are correct in this context
03370       BEGIN
03380        LOCAL c;
03390        id[ w1 ]<lsby> = .ch;
03400        read( prr , c ); id[ w1 ]<msby> = .c;
03410        read( prr , c ); id[ w2 ]<lsby> = .c;
03420        ! id[ w2 ]<msby> is SPACE always
03430        IF .listing THEN
03440         BEGIN
03450          INCR i FROM 0 TO 3 DO
03460           write(lstfil,space);
03470          c = .id[ w1 ]<lsby>;
03480          write(lstfil,.c);
03490 !         %debug% putch(.ttyno,.c);
03500          c = .id[ w1 ]<msby>;
03510          write(lstfil,.c);
03520 !         %debug% putch(.ttyno,.c);
03530          c = .id[ w2 ]<lsby>;
03540          write(lstfil,.c)
03550 !         %debug% ;putch(.ttyno,.c)
03560         END;
03570        IF NOT endofline( prr ) THEN
03580          BEGIN
03590           read( prr , ch );
03600 !          %debug% putch(.ttyno,.ch);
03610           IF .listing THEN write(lstfil,.ch)
03620          END;
03630       END; ! getid
03640
03650     MACRO maccomp(oper,table)=
03660       .(id[w1]+.q)<0,8> OPER .(table[.k,w1]+.q)<0,8> $;
03670
03680     ROUTINE compare=
03690      BEGIN
03700       q = 0; p = true;
03710       WHILE .p AND ( .q LEQ 3 ) DO
03720        IF maccomp(EQL,inst) THEN q = .q + 1
03730         ELSE p = false;
03740      END; ! compare
03750
03760      ROUTINE legcomp=
03770      BEGIN
03780       compare();
```

```
03790          RETURN (( maccomp(LEQ,inst)) OR .p )
03800         END; ! leqcomp
03810
03820         ROUTINE geqcomp=
03830          BEGIN
03840           compare();
03850           RETURN (( maccomp(GEQ,inst)) OR .p)
03860          END; ! geqcomp
03870
03880         ROUTINE neqcomp=
03890         BEGIN
03900          q = 0; p = true;
03910          WHILE .p AND ( .q LEQ 3 ) DO
03920           IF maccomp(EQL,sptab) THEN q = .q + 1
03930            ELSE p = false;
03940          RETURN NOT .p;
03950         END; ! neqcomp
03960
03970         ROUTINE binarysearch=
03980          BEGIN
03990           LOCAL i , j;
04000           i = 0; j = nois;
04010           DO
04020            BEGIN
04030             k = ( .i + .j ) DIV 2;
04040     !%debug%putlch(.ttyno,.inst[.k,w1]<lsby>);putlch(.ttyno,.inst[.k,w1]<msby>);
04050     !%debug%putlch(.ttyno,.inst[.k,w2]<lsby>);putlch(.ttyno,.inst[.k,w2]<msby>);ttcrlf(.ttyno);
04060             IF leqcomp() THEN j = .k - 1;
04070             IF geqcomp() THEN i = .k + 1;
04080            END
04090           UNTIL .i GTR .j;
04100           IF .i = 1 GTR .j THEN op = .opc[ .k ]
04110            ELSE asserr( 3 );
04120          END; ! binarysearch
04130
04140         ROUTINE relopinsts =
04150          BEGIN
04160           IF .ch EQL aye OR .ch EQL eye THEN k = .op + 16
04170            ELSE IF .ch EQL are THEN k = .op + 43
04180            ELSE IF .ch EQL yes THEN k = .op + 27
04190            ELSE IF .ch EQL emm THEN
04200                  BEGIN
04210                   k = .op + 95;
04220                   readint( prr , q );
04230                  END
04240                  ELSE ! .ch EQL bee
04250                   k = .op;
04260           loadbyte( .k );
04270           IF .ch EQL emm THEN loadword( .q );
04280          END; ! relopinsts
04290
04300         ROUTINE noadchinsts =
04310          BEGIN
04320           loadbyte( .op );
```

```
04330           readint( prr , q );
04340           loadword( .q );
04350         END; ! noaddrinsts
04360
04370       ROUTINE entjumps =
04380         BEGIN
04390           loadbyte( .op );
04400           labelsearch();
04410         END; ! entjumps
04420
04430       ROUTINE oneopldst =
04440         BEGIN
04450           k = ( IF .ch EQL are THEN .op + 1
04460                 ELSE IF .ch EQL yes THEN .op + 2
04470                 ELSE .op % A,B,I % );
04480           loadbyte( .k );
04490           readint( prr , q );
04500           loadword( .q );
04510         END; ! oneopldst
04520
04530       ROUTINE twoopldst =
04540         BEGIN
04550           k = ( IF .ch EQL are THEN .op + 1
04560                 ELSE IF .ch EQL yes THEN .op + 2
04570                 ELSE .op % A,B,I ,space % );
04580           loadbyte( .k );
04590           readint( prr , p );
04600           readint( prr , q );
04610           loadbyte( .p );
04620           loadword( .q );
04630         END; ! twoopldst
04640
04650         ! main program of assembler
04660       getid();
04670       binarysearch();
04680       SELECT .op OF
04690         NSET
04700           CSPP : BEGIN
04710                    WHILE .ch EQL space DO read( prr , ch );
04720                    getid(); k = 0; ! linear search of SPTABLE
04730                    WHILE neqcomp() DO k = .k + 1;
04740                     ! NOTE : no boundary checks
04750                    loadbyte( .k );
04760                  END;
04770           CHKP : BEGIN
04780                    loadbyte( .op );
04790                    INCR i FROM 0 TO 1 DO
04800                      BEGIN
04810                        readint( prr , q ); loadword( .q );
04820                      END
04830                  END;
04840           LOCP : BEGIN
04850                    IF .ch EQL eye THEN
04860                      BEGIN
```

```
04870                   loadbyte( LDCIP );
04880                   readint( prr , q );
04890                   loadword( .q );
04900                  END  ! T
04910                 ELSE IF .ch EQL are THEN asserr( 4 ) ! R
04920                 ELSE IF .ch EQL yen THEN loadbyte( LDCNP ) ! N
04930                 ELSE IF .ch EQL bee THEN
04940                       BEGIN
04950                        readint( prr , p );
04960                        loadbyte(( IF .p THEN LDCBTP ELSE LDCBFP ));
04970                       END ! B
04980                  ELSE IF .ch EQL lpar THEN
04990                       BEGIN
05000                        LOCAL VECTOR setval[ 4 ];
05010                        loadbyte( LDCSP );
05020                        INCR i FROM 0 TO 3 DO setval[ .i ] = 0;
05030                        read( prr , ch );
05040                        WHILE .ch NEQ rpar DO
05050                         BEGIN
05060                          readint( prr , q );
05070                          setval[ .q DIV 16 ] = .setval[ .q DIV 16 ]
05080                                                OR ( 1 ^ ( .q MOD 16 ) );
05090                          read( prr , ch );
05100                         END;
05110                        INCR i FROM 0 TO 3 DO loadword( .setval[ .i ] );
05120                       END; ! S
05130           END; ! LDCP
05140   CUPP : BEGIN
05150           loadbyte( .op );
05160           readint( prr , p ); loadbyte( .p );
05170           labelsearch();
05180          END;
05190   MSTP : BEGIN
05200           loadbyte( .op );
05210           readint( prr , p );
05220           loadbyte( .p )
05230          END;
05240   RETP : BEGIN
05250           loadbyte( .op );
05260           p = ( IF .ch EQL eye THEN 1
05270                 ELSE IF .ch EQL are THEN 2
05280                 ELSE IF .ch EQL sea THEN 3
05290                 ELSE IF .ch EQL bee THEN 4
05300                 ELSE IF .ch EQL aye THEN 5
05310                 ELSE % .ch EQL pee % 0 );
05320           loadbyte( prr , .p );
05330          END; ! RETP
05340   LCAP : BEGIN
05350           loadbyte( .op );
05360           read( prr , ch );
05370              ! .ch is the first character of the string
05380           WHILE .ch NEQ quote DO
05390            BEGIN
05400             loadbyte( .ch );
```

```
05410                   read( prr , ch );
05420                  END;
05430                 loadbyte( nul );
05440                END; ! LCAP
05450        STOP : BEGIN    ! STO - Pinstruction
05460                k = ( IF .ch EOL are THEN .op + 1
05470                      ELSE IF .ch EOL yes THEN .op + 2
05480                      ELSE .op % A,B,I % );
05490                 loadbyte( .k );
05500                END;
05510        LESP : relopinsts();
05520        LEQP : relopinsts();
05530        EQUP : relopinsts();
05540        NEQP : relopinsts();
05550        GEQP : relopinsts();
05560        GRTP : relopinsts();
05570        DECP : noadchinsts();
05580        INCP : noadchinsts();
05590        TXAP : noadchinsts();
05600        LALP : noadchinsts();
05610        LADP : noadchinsts();
05620        MOVP : noadchinsts();
05630        FJPP : entjumps();
05640        XJPP : entjumps();
05650        UJPP : entjumps();
05660        ENTP : entjumps();
05670        INDP : oneopldst();
05680        LDLP : oneopldst();
05690        LDOP : oneopldst();
05700        STLP : oneopldst();
05710        SROP : oneopldst();
05720        LODP : twoopldst();
05730        STRP : twoopldst();
05740        LDAP : twoopldst();
05750        OTHERWISE : loadbyte( .op );
05760       TESN;
05770      readln( prr );
05780     END; ! assembler
05790
05800    GLOBAL ROUTINE asm( lstn ) =
05810     BEGIN
05820      LOCAL maxppc;
05830      listing = .lstn;
05840      ttcrlf(.ttyno); ttoutstr(.ttyno,version);
05850      ttcrlf(.ttyno);
05860      initassembler();
05870      generate();
05880      maxppc = .ppc - 1;
05890      ppc = 0;
05900      read( prr , ch );
05910      generate();
05920      fileclose(prr);
05930      IF .listing THEN fileclose(lstfil);
05940      ttoutstr(.ttyno,mes1); ttcrlf(.ttyno);
```

```
05950          RETURN .maxppc
05960        END; ! asm
05970
05980      END
05990    ELUDOM
```

```
00010       %
00020       ************************************************************************
00030       **********                                                    **********
00040       **********   MODULE : PMACHINE                                **********
00050       **********   DATE : 1 JULY 1982                               **********
00060       **********   AUTHOR : S SRINIVAS RAGHURAM                     **********
00070       **********   INPUT : P-Machine's object code in P-Machine's   **********
00080       **********           PCODE array.                             **********
00090       **********   FILES : Files INPUT & PRD are input files.       **********
00100       **********           and OUTPUT & PRR are the output files    **********
00110       **********           produced by the interpreter              **********
00120       **********   ENTRY : INTR(TRCE,TTYOUT);                       **********
00130       **********           trce=true then execution trace is        **********
00140       **********           generated.                               **********
00150       **********           ttyout=true then the results being       **********
00160       **********           written on the file output are also      **********
00170       **********           displayed on the terminal.               **********
00180       **********                                                    **********
00190       ************************************************************************%
00200       MODULE pmachine ( unsafe , nooptimize ,nofinal ,   nolist   ) =
00210             %     P - Machine ERROKS:
00220                     0  : HEAP OVERFLOWS STACK
00230                     1  : ILLEGAL INSTRUCTION
00240                     2  : NOT IMPLEMENTED
00250                     3  : ERROR IN CASE STATEMENT
00260                     4  : STACK OVERFLOWS HEAP
00270                     5  : VALUE OUT OF RANGE
00280                     6  : . . . . . .
00290                                                              %
00300
00310        BEGIN
00320         MACRO
00330          t1bit    =  7,1    $,    ! type1 or type2 instruction if set else type0
00340          t2bit    =  6,1    $,    ! type2 instruction if set
00350          bits6    =  0,6    $,    ! least significant six bits
00360          lsby     =  0,8    $,    ! least significant byte
00370          msby     =  8,8    $,    ! most  significant byte
00380          bool     =  0,1    $,    !
00390          sign     = 15,1    $;
00400
00410         MACRO
00420          cw = pstk[ .psp - 1 ] $,      ! current word pointed by sp
00430
00440          nw = pstk[ .psp + 1 ] $,      ! next word
00450
00460          pw = pstk[ .psp - 3 ] $,      ! previous word
00470
00480          pushi = psp = .psp + 2 $,     ! push integer
00490
00500          popi  = psp = .psp - 2 $,     ! pop integer
00510
00520          pops  = psp = .psp - 8 $;     ! pop set ( 4 words )
00530
00540         BIND
```

```
00550         version = UPLIT ASCIZ "PAS316 P-Interpreter Version of Jul 1,82",
00560         mes1    = UPLIT ASCIZ "End of Execution",
00570         err0    = UPLIT ASCIZ "HEAP OVERRUNS STACK",
00580         err1    = UPLIT ASCIZ "ILLEGAL INSTRUCTION",
00590         err2    = UPLIT ASCIZ "NOT IMPLEMENTED",
00600         err3    = UPLIT ASCIZ "ERROR IN CASE STATEMENT",
00610         err4    = UPLIT ASCIZ "STACK OVERRUNS HEAP",
00620         err5    = UPLIT ASCIZ "VALUE OUT OF RANGE";
00630
00640
00650       BIND
00660         maxcode   =  #40000  ,     ! size of code array
00670         maxstk    =  #40000  ,     ! size of PASCAL STACK
00680         nilvalue  =  #40000  ,
00690         expstk    =  #100    ,     ! max size by which STACK can grow
00700                                    ! during expression evaluation
00710         false     =   0      ,
00720         true      =   1      ;
00730         outadr    =  12      ;
00740         ! ASCII codes
00750         zero = #60,  nine = #71,  nul = #0,  space = #40,  plus = #53,  minus = #55;
00760
00770       EXTERNAL reset , rewrite;
00780       EXTERNAL get , put;
00790       EXTERNAL read , readln , readint , write , writeln;
00800       EXTERNAL endoffile , endofline;
00810       EXTERNAL allclose;
00820       EXTERNAL ttcrlf , ttputoct , ttoutstr , putlch;
00830       EXTERNAL fetch;
00840       EXTERNAL getfilename;
00850       EXTERNAL ttyno;
00860
00870       STRUCTURE pmcstack [ i ] = [ i * BYTES ]
00880         ( .pmcstack + .i * BYTES ) <0,16>;
00890
00900       BYTE OWN pmcstack pstk [ maxstk ];
00910
00920
00930       BYTE OWN VECTOR string [ 16 ];
00940
00950       ! P-MACHINE registers
00960
00970       OWN   pnc   ,     ! Program Counter
00980             psp   ,     ! Stack Pointer : points to top of pstk
00990             mp    ,     ! Mark Pointer  : points to beginning of a data segment
01000             np    ,     ! New Pointer   : points to top of dynamically allocated area
01010             p     ,     ! INSTRUCTION REGISTER : P - field
01020             q     ,     ! INSTRUCTION REGISTER : Q - field
01030             l     ,     ! GPR : Low register
01040             c     ,     ! GPR : Count register
01050             b     ;     ! GPR : Boolean register
01060
01070       BYTE OWN op ,     ! INSTRUCTION REGISTER : OPCODE - field
01080             interpret;  ! controls execution of interpreter
```

```
01090
01100        OWN VECTOR cb [ 4 ] ;    ! Constant Buffer
01110
01120        OWN  trace, outtty;
01130
01140         ROUTINE pmcerr( n ) =
01150         BEGIN
01160          LOCAL errmesg;
01170          ttcrlf(.ttyno);
01180          ttoutstr(.ttyno,UPLIT ASCIZ "Run Time ERROR: ");
01190          ttputoct(.ttyno,.n);
01200          ttcrlf(.ttyno);
01210          CASE .n OF
01220           SET
01230            errmesg = err0;
01240            errmesg = err1;
01250            errmesg = err2;
01260            errmesg = err3;
01270            errmesg = err4;
01280            errmesg = err5
01290           TES;
01300          ttoutstr(.ttyno,.errmesg); ttcrlf(.ttyno);
01310          ttoutstr(.ttyno,UPLIT ASCIZ " PPC = "); ttputoct(.ttyno,.ppc-1);
01320          ttoutstr(.ttyno,UPLIT ASCIZ "  UP = "); ttputoct(.ttyno,.op);
01330          ttoutstr(.ttyno,UPLIT ASCIZ " PSP = "); ttputoct(.ttyno,.psp-1);
01340          ttoutstr(.ttyno,UPLIT ASCIZ "  MP = "); ttputoct(.ttyno,.mp);
01350          ttoutstr(.ttyno,UPLIT ASCIZ "  NP = "); ttputoct(.ttyno,.np);
01360          ttcrlf(.ttyno);
01370          EMT(0)
01380         END; ! pmcerr
01390
01400        ROUTINE locfetch =
01410         BEGIN
01420          BYTE LOCAL tmp;
01430          tmp = fetch(.ppc); ppc = .ppc + 1;
01440          RETURN .tmp;
01450         END; ! locfetch
01460
01470        ROUTINE qlocfetch =
01480         BEGIN
01490          q<lsby> = locfetch();
01500          q<msby> = locfetch();
01510         END; ! qlocfetch
01520
01530        ROUTINE base( p ) =
01540         BEGIN
01550          STACKLOCAL ad;
01560           ad = .mp;
01570           WHILE .p GTR 0 DO
01580            BEGIN
01590             ad = .pstk[ .ad + 4 ];  ! static link
01600             p = .p - 1;
01610            END;
01620           RETURN .ad;
```

```
01630          END; ! base
01640
01650          ROUTINE filadr =
01660           ( .cw - 10 ) DIV 2 ;
01670
01680          ROUTINE writestring( f , l , h ) =
01690           BEGIN
01700            IF .h GTR .l THEN
01710             INCR i FROM 1 TO ( .h - .l ) DO
01720              BEGIN
01730               IF .outtty THEN IF .cw EQL outadr THEN
01740                putlch(.ttyno,space);
01750               write( .f , space )
01760              END
01770            ELSE l = .h;
01780            INCR i FROM 0 TO ( .l - 1 ) DO
01790             BEGIN
01800              IF .outtty THEN IF .cw EQL outadr THEN
01810               putlch(.ttyno,.string[.i]);
01820              write( .f , .string[ .i ])
01830             END;
01840           END; ! writestring
01850
01860          ROUTINE coctintstr( n ) =
01870           BEGIN
01880            LOCAL dg;
01890            dg = .n<sign>; string[0] = zero + .dg;
01900            INCR i FROM 1 TO 5 DO
01910             BEGIN
01920              dg = .n<12,3>;
01930              string[ .i ] = zero + .dg;
01940              n = .n * 8;
01950             END;
01960            INCR i FROM 6 TO 9 DO string[ .i ] = space;
01970           END; ! coctintstr
01980
01990          ROUTINE cdecintstr( n ) =
02000           BEGIN
02010            BYTE LOCAL VECTOR stk[ 5 ];
02020            BYTE LOCAL ind , sgn;
02030            sgn = .n<sign>;
02040            string[0] =( IF .sgn THEN minus ELSE space );
02050            IF .sgn THEN n = -.n;
02060            ind = -1;
02070            DO
02080             (ind = .ind + 1;
02090             stk[ .ind ] = .n MOD 10;
02100             n = .n DIV 10;)
02110            UNTIL .n EQL 0;
02120            INCR i FROM .ind+2 TO 9 DO string[ .i ] = space;
02130            sgn = 1;  ! henceforth used as array index for string
02140            DECR i FROM .ind TO 0 DO
02150             BEGIN
02160              string[ .sgn ] = zero + .stk[ .i ];
```

```
02170            sgn = .sgn + 1;
02180           END;
02190         END; ! cdecintstr
02200
02210        ROUTINE compare =
02220         BEGIN
02230         pop1;
02240         p = .cw; l = .nw;
02250         c = 0; b = true;
02260         WHILE .b AND ( .c NEQ .q ) DO
02270          IF .pstk[ .p + .c ] EQL .pstk[ .l + .c ] THEN c = .c + 2
02280          ELSE b = false;
02290         END; ! compare
02300
02310       ROUTINE execone =
02320        BEGIN ! type0 instructions : standard proc/func & no operand instructions
02330         MACRO relop( WHAT ) =
02340           BEGIN
02350            pop1;
02360            cw = .cw<bool> WHAT .nw<bool>;
02370           END$ ;
02380
02390         MACRO intgop( what )=
02400          BEGIN
02410           pop1;
02420           cw = .cw WHAT .nw;
02430          END$ ;
02440
02450         MACRO b1 = 1 $ , b2 = c $ ;
02460
02470         MACRO setop( WHAT ) =
02480           BEGIN
02490            pops;
02500            b1 = .psp - 7; b2 = .psp + 1;
02510            INCR i FROM 0 TO 3 DO
02520            pstk[ .b1 + 2 * .i ] = .pstk[ .b1 + 2 * .i ] WHAT .pstk[ .b2 + 2 * .i ];
02530           END$ ;
02540
02550       ROUTINE ex10 =     ! standard procedure/function
02560        BEGIN
02570         LOCAL opcode;
02580         opcode = .op;
02590         CASE .opcode OF
02600          SET
02610
02620           ! SPECIAL PROCEDURES / FUNCTIONS
02630           ! 0 : ELN
02640           cw = endofline( filadr() );
02650
02660           ! 1 : GET
02670           BEGIN
02680            get( filadr() ); pop1;
02690           END;
02700
```

```
02710           ! 2 : RDC
02720           BEGIN
02730            read( filadr() , pstk[ .psp ] );
02740            psp = .psp - 4
02750           END;
02760
02770           ! 3 : RDI
02780           BEGIN
02790            readint( filadr() , pstk[ .psp ]  );
02800            psp = .psp - 4
02810           END;
02820
02830           ! 4 : RDR
02840           omcerr(2);
02850
02860           ! 5 : RLN
02870           BEGIN
02880            readln( filadr() ); popi;
02890           END;
02900
02910           ! 6 : PUT
02920           BEGIN
02930            put( filadr() ); popi;
02940           END;
02950
02960           ! 7 : WRC
02970           BEGIN
02980            string[0] = .pstk[ .psp - 5 ];
02990            writestring( filadr(), 1 , .pw);
03000            psp = .psp - 6;
03010           END;
03020
03030           ! 8 : WRI
03040           BEGIN
03050            cdecintstr( .pstk[ .psp - 5 ] );
03060            writestring( filadr() , 10 , .pw );
03070            psp = .psp - 6;
03080           END;
03090
03100           ! 9 : WRO
03110           BEGIN
03120            octintstr( .pstk[ .psp - 5 ] );
03130            writestring( filadr() , 10 , .pw );
03140            psp = .psp - 6;
03150           END;
03160
03170           !10 : WRR
03180           omcerr(2);
03190
03200           !11 : WRS
03210           BEGIN
03220            writestring( filadr() , .pstk[ .psp - 5 ] , .pw );
03230            psp = .psp - 6;
03240           END;
```

```
03250
03260           !12 : WLN
03270           BEGIN
03280            writeln( filadr() );
03290            IF .outtty THEN IF .cw EQL outadr THEN
03300             ttcrlf(.ttyno);
03310            popi;
03320           END;
03330
03340           !13 : NEW
03350           BEGIN
03360            STACKLOCAL ad;
03370            ad = .np - .cw;
03380            IF .ad LEQ ( .psp + expstk ) THEN pmcerr(0);
03390             ! HEAP overflows STACK
03400            np = .ad;
03410            ad = .pw;
03420            pstk[ .ad ] = .np;
03430            psp = .psp - 4;
03440           END;
03450
03460           !14 : PAK
03470           pmcerr(2);
03480
03490           !15 : RST
03500           BEGIN
03510            np = .cw;
03520            popi;
03530           END;
03540
03550           !16 : SAV
03560           BEGIN
03570            pstk[ .cw] = .np; popi;
03580           END;
03590
03600           !17 : SIN
03610           pmcerr(2);
03620
03630           !18 : COS
03640           pmcerr(2);
03650
03660           !19 : ATN
03670           pmcerr(2);
03680
03690           !20 : EXP
03700           pmcerr(2);
03710
03720           !21 : LOG
03730           pmcerr(2);
03740
03750           !22 : SQT
03760           pmcerr(2);
03770
03780           !23 : UNUSED OPCODE
```

```
03790           ; ! nooperation
03800
03810           !24 : UNUSED OPCODE
03820           ; ! nooperation
03830
03840           !25 : UNUSED OPCODE
03850           ; ! nooperation
03860
03870           !26 : UNUSED OPCODE
03880           ; ! nooperation
03890
03900           !27 : UNUSED OPCODE
03910           ; ! nooperation
03920
03930           !28 : UNUSED OPCODE
03940           ; ! nooperation
03950
03960           !29 : UNUSED OPCODE
03970           ; ! nooperation
03980
03990           !30 : UNUSED OPCODE
04000           ; ! nooperation
04010
04020           !31 : UNUSED OPCODE
04030           ; ! nooperation
04040         TES;
04050       END;    ! ex10
04060
04070     ROUTINE ex11 =      ! boolean and integer operations
04080      BEGIN
04090       LOCAL opcode;
04100       opcode = .op - 32;
04110       CASE .opcode OF
04120        SET
04130
04140         ! NO OPERAND INSTRUCTIONS
04150
04160         !32 : EOF
04170         cw = endoffile( filadr() );
04180
04190         !33 : NOT
04200         cw = NOT .cw;
04210
04220         !34 : IOR
04230         BEGIN
04240          pop1;
04250          cw = .cw OR .nw;
04260         END;
04270
04280         !35 : AND
04290         BEGIN
04300          pop1;
04310          cw = .cw AND .nw;
04320         END;
```

```
04330
04340           !36: LDCBF
04350           BEGIN
04360            nw = false; pushi;
04370           END;
04380
04390           !37: LDCBT
04400           BEGIN
04410            nw = true; pushi;
04420           END;
04430
04440           !38: LESB
04450           relop( LSS );
04460
04470           !39: LEQB
04480           relop( LEQ );
04490
04500           !40: EQUB
04510           relop( EQL );
04520
04530           !41: NEQB
04540           relop( NEQ );
04550
04560           !42: GEQB
04570           relop( GEQ );
04580
04590           !43: GRTB
04600           relop( GTR );
04610
04620           !44: ODD
04630           cw = .cw<bool>;
04640
04650           !45: NGI
04660           cw = -.cw;
04670
04680           !46: ADI
04690           intgop( + );
04700
04710           !47: SBI
04720           intgop( - );
04730
04740           !48: MPI
04750           intgop( * );
04760
04770           !49: DVI
04780           intgop( DIV );
04790
04800           !50: MOD
04810           intgop( MOD );
04820
04830           !51: SQI
04840           cw = .cw * .cw;
04850
04860           !52: ABI
```

```
04870           IF .cw<sign> THEN cw = -.cw;
04880
04890           !53: LDCN
04900           BEGIN
04910            nw = nilvalue; pushi;
04920           END;
04930
04940           !54: LESA,LESI
04950           intgop( LSS );
04960
04970           !55: LEQA,LEQI
04980           intgop( LEQ );
04990
05000           !56: EQUA,EQUI
05010           intgop( EQL );
05020
05030           !57: NEQA,NEQI
05040           intgop( NEQ );
05050
05060           !58: GEQA,GEQI
05070           intgop( GEQ );
05080
05090           !59: GRTA,GRTI
05100           intgop( GTR );
05110          TES;
05120        END;    ! ex11
05130
05140      ROUTINE ex12 =     ! set operations
05150       BEGIN
05160        LOCAL opcode;
05170        opcode = .op - 60;
05180        CASE .opcode OF
05190         SET
05200
05210           !60: SGS
05220           BEGIN
05230            c = .cw;
05240            cw = nw = pstk[ .psp + 3 ] = pstk[ .psp + 5 ] = 0;
05250             pstk[ .psp - 1 + 2 * ( .c DIV 16 ) ] = 1 ^ ( .c MOD 16 );
05260            psp = .psp + 6;
05270           END;
05280
05290           !61: INN
05300           BEGIN
05310            pops;
05320            cw = ( IF ( 1 ^ ( .cw MOD 16 ) AND
05330                      .pstk[ .psp + 1 + 2 * ( .cw DIV 16 ) ] )  EOL 0 THEN false
05340                  ELSE true );
05350           END;
05360
05370           !62: UNI                    A = A OR B
05380           setop( OR );
05390
05400           !63: INT                    A = A AND B
```

```
05410           setop( AND );
05420
05430           !64: DIF                    A = A AND NOT B
05440           setop( AND NOT );
05450
05460           !65: LESS
05470           pmcerr(1);
05480
05490           !66: LEQS       A IS A SUBSET OF B. IF A = (A AND B)
05500           BEGIN
05510            psp = .psp - 14;
05520            b1 = .psp - 1; b2 = .psp + 7;
05530            INCR i FROM 0 TO 3 DO
05540             BEGIN
05550              pstk[ .b1 ] = ( .pstk[ .b1 + 2 * .i ] EQL ( .pstk[ .b1 + 2 * .i ]
05560                                AND .pstk[ .b2 + 2 * .i ] ) );
05570              IF NOT .pstk[ .b1 ] THEN EXITLOOP;
05580             END
05590           END;
05600
05610           !67: EQUS
05620           BEGIN
05630            psp = .psp - 14;
05640            b1 = .psp - 1; b2 = .psp + 7;
05650             INCR i FROM 0 TO 3 DO
05660             BEGIN
05670              pstk[ .b1 ] = .pstk[ .b1 + 2 * .i ] EQL .pstk[ .b2 + 2 * .i ];
05680              IF NOT .pstk[ .b1 ] THEN EXITLOOP;
05690             END
05700           END;
05710
05720           !68: NEQS
05730           BEGIN
05740            psp = .psp - 14;
05750            b1 = .psp - 1; b2 = .psp + 7;
05760            INCR i FROM 0 TO 3 DO
05770             BEGIN
05780              pstk[ .b1 ] = .pstk[ .b1 + 2 * .i ] NEQ .pstk[ .b2 + 2 * .i ];
05790              IF NOT .pstk[ .b1 ] THEN EXITLOOP;
05800             END
05810           END;
05820
05830           !69: GEQS     B IS A SUBSET OF A . IF B = ( B AND A )
05840           BEGIN
05850            psp = .psp - 14;
05860            b1 = .psp - 1; b2 = .psp + 7;
05870            INCR i FROM 0 TO 3 DO
05880             BEGIN
05890              pstk[ .b1 ] = ( .pstk[ .b2 + 2 * .i ] EQL
05900                            ( .pstk[ .b2 + 2 * .i ] AND .pstk[ .b1 + 2 * .i ] ) );
05910              IF NOT .pstk[ .b1 ] THEN EXITLOOP;
05920             END
05930           END;
05940
```

```
05950           !70 : GRTS
05960           omcerr(1);
05970         TES;
05980       END;    !ex12
05990
06000     ROUTINE ex13 =  ! real and rest of the operations
06010      BEGIN
06020       LOCAL opcode;
06030       opcode = .op - 71;
06040       CASE .opcode OF
06050        SET
06060
06070          !71: TRC
06080          omcerr(2);
06090
06100          !72: FLO
06110          omcerr(2);
06120
06130          !73: FLT
06140          omcerr(2);
06150
06160          !74: NGR
06170          omcerr(2);
06180
06190          !75: ADR
06200          omcerr(2);
06210
06220          !76: SBR
06230          omcerr(2);
06240
06250          !77: MPR
06260          omcerr(2);
06270
06280          !78: DVR
06290          omcerr(2);
06300
06310          !79: SQR
06320          omcerr(2);
06330
06340          !80: ABR
06350          omcerr(2);
06360
06370          !81: LESR
06380          omcerr(2);
06390
06400          !82: LEQR
06410          omcerr(2);
06420
06430          !83: EQUR
06440          omcerr(2);
06450
06460          !84: NEQR
06470          omcerr(2);
06480
```

```
06490           !85: GEQR
06500           pmcerr(2);
06510
06520           !86: GRTR
06530           pmcerr(2);
06540
06550           !87 : UJC
06560           pmcerr(3);
06570
06580           !88: STP
06590           interpret = false;
06600         TES;
06610        END;    ! ex13
06620       IF .op LEQ 31 THEN ex10()
06630        ELSE IF .op LEQ 59 THEN ex11()
06640        ELSE IF .op LEQ 70 THEN ex12()
06650        ELSE ex13();
06660       END; ! execone
06670
06680      ROUTINE exectwo =
06690       BEGIN
06700      ROUTINE ex20 =  ! load store and block rel ops.
06710       BEGIN
06720        LOCAL opcode;
06730        opcode = .op - 128;
06740        CASE .opcode OF
06750         SET
06760          !128: LDC
06770          BEGIN
06780           INCR i FROM 0 TO .c DO
06790            pstk[ .psp + 1 + 2 * .i ] = .cb[ .i ];
06800           psp = .psp + 2 * ( .c + 1 );
06810          END;
06820
06830          !129: IND
06840          BEGIN
06850           q = .cw + .q;
06860           INCR i FROM 0 TO .c DO
06870            pstk[ .psp - 1 + 2 * .i ] = .pstk[ .q + 2 * .i ];
06880           psp = .psp + 2 *  .c;
06890          END;
06900
06910          !130: LDL,LDO,LOD
06920          BEGIN
06930           INCR i FROM 0 TO .c DO
06940            pstk[ .psp + 1 + 2 * .i ] = .pstk[ .q + 2 * .i ];
06950           psp = .psp + 2 * ( .c + 1 );
06960          END;
06970
06980          !131: STO
06990          BEGIN
07000           LOCAL locpsp;
07010           locpsp = .psp - 2 * ( .c + 1 ) - 2;
07020           psp = .locpsp;
```

```
07030             q = .nw;
07040             INCR i FROM 0 TO .c DO
07050               pstk[ .q + 2 * .i ] = .pstk[ .osp + 3 + 2 * .i ];
07060           END;
07070
07080           !132: STL,SRO,STR
07090           BEGIN
07100             LOCAL locpsp;
07110             locpsp = .psp - 2 * ( .c + 1 );
07120             psp = .locpsp;
07130             INCR i FROM 0 TO .c DO
07140               pstk[ .q + 2 * .i ] = .pstk[ .psp + 1 + 2 * .i ];
07150           END;
07160
07170           !133: LESM
07180           BEGIN
07190             compare();
07200             cw = ( .pstk[ .p + .c ] LSS .pstk[ .i + .c ] ) AND NOT .b;
07210           END;
07220
07230           !134: LEQM
07240           BEGIN
07250             compare();
07260             cw = ( .pstk[ .p + .c ] LEQ .pstk[ .i + .c ] ) OR .b;
07270           END;
07280
07290           !135: EQUM
07300           BEGIN
07310             compare();
07320             cw = .b;
07330           END;
07340
07350           !136: NEQM
07360           BEGIN
07370             compare();
07380             cw = NOT .b;
07390           END;
07400
07410           !137: GEQM
07420           BEGIN
07430             compare();
07440             cw = ( .pstk[ .p + .c ] GEQ .pstk[ .i + .c ] ) OR .b;
07450           END;
07460
07470           !138: GRTM
07480           BEGIN
07490             compare();
07500             cw = ( .pstk[ .p + .c ] GTR .pstk[ .i + .c ] ) AND NOT .b;
07510           END;
07520         TES;
07530       END;    !   ex20
07540
07550
07560     ROUTINE ex21 =
```

```
08110            psp = .psp - 4;
08120            INCR i FROM 0 TO .q - 1 BY 2 DO
08130             pstk[ .p + .i ] = .pstk[ .i + .i ];
08140           END;
08150
08160           !149: LDCI
08170           BEGIN
08180            nw = .q; pushi;
08190           END;
08200
08210           !150: LDA
08220           BEGIN
08230            nw = base( .p ) + .q;
08240            pushi;
08250           END;
08260
08270           !151: CUP
08280           BEGIN
08290            mp = .psp - ( .p + 9 );
08300            pstk[ .mp + 8 ] = .ppc;
08310            ppc = .q;
08320           END;
08330
08340           !152: MST
08350           BEGIN
08360            pstk[ .psp + 5 ] = base( .p );
08370            pstk[ .psp + 7 ] = .mp;
08380            psp = .psp + 10;
08390           END;
08400
08410           !153: RET
08420           BEGIN
08430            psp = ( IF .p EQL 0 THEN .mp - 1
08440                    ELSE IF .p EQL 2 THEN .mp + 3
08450                         ELSE .mp + 1 );
08460            ppc = .pstk[ .mp + 8 ];
08470            mp = .pstk[ .mp + 6 ];
08480           END;
08490
08500           !154: CHK
08510           IF ( .cw LSS .1 ) OR ( .cw GTR .q ) THEN pmcerr(5);
08520
08530           !155: LCA
08540           ; ! NO OPERATION
08550
08560          TES;
08570         END;  ! ex21
08580        IF .op LEQ 138 THEN ex20()
08590         ELSE ex21();
08600        END; ! exectwo
08610
08620       ROUTINE type1 =
08630        BEGIN
08640         IF .op LEQ 149 THEN     ! group 3
```

```
08650          ! LESM,LEOM,EQUM,NEOM,GEQM,GRTM,DEC,INC,
08660          ! IXA ,LAL ,LAO ,FJP ,XJP ,UJP ,ENT,MOV,LDCI
08670          qlocfetch()
08680        ELSE IF .op LEQ 151 THEN  ! group 4
08690                BEGIN ! LDA,CUP
08700                 p = 0;
08710                  p<lsby> = locfetch();
08720                 qlocfetch();
08730                END
08740        ELSE IF .op LEQ 153 THEN  ! group 2
08750                BEGIN ! MST,RET
08760                 p = 0;
08770                 p<lsby> = locfetch();
08780                END
08790        ELSE IF .op EQL 154 THEN  ! group 5
08800                BEGIN ! CHK
08810                 l<lsby> = locfetch();
08820                 l<msby> = locfetch();
08830                 qlocfetch();
08840                END
08850        ELSE IF .op EQL 155 THEN  ! group 6
08860                BEGIN ! LCA
08870                 C = 0 ; P = 0;
08880                 p<lsby> = locfetch();
08890                 WHILE .p<lsby> NEQ nul DO
08900                  BEGIN
08910                   string[ .c ] = .p<lsby>;
08920                   c = .c + 1;
08930                   p<lsby> = locfetch();
08940                  END;
08950                END;
08960          exectwo();
08970      END; ! type1
08980
08990      ROUTINE type2execzero =
09000       BEGIN
09010        MACRO
09020         m0( count , nop ) =
09030          BEGIN
09040           c = count; op = nop;
09050          END $ ,
09060
09070         m1( count , nop ) =
09080          BEGIN
09090           c = count; qlocfetch(); op = nop;
09100          END $ ,
09110
09120         m2( count , nop ) =
09130          BEGIN
09140           c = count; qlocfetch(); q = .mp + .q; op = nop;
09150          END $ ,
09160
09170         m3( count , nop ) =
09180          BEGIN
```

```
09190            c = count; p = 0; p<lsby> = locfetch(); qlocfetch();
09200            q = base( .p ) + .q;
09210            op = nop;
09220           END $ ,
09230
09240          m4( count , nop ) =
09250           BEGIN
09260            INCR i FROM 0 TO count DO
09270             BEGIN
09280              cb[ .i ]<lsby> = locfetch();
09290              cb[ .i ]<msby> = locfetch();
09300             END;
09310            c = count; op = nop;
09320           END $;
09330
09340         CASE .op<bits6> OF
09350          SET
09360            !192: LDCR  group 6
09370            m4( 1 , 128 );
09380            !193: LDCS
09390            m4( 3 , 128 );
09400            !194: INDA,INDI,INDB  group 3
09410            m1( 0 , 129 );
09420            !195: INDR
09430            m1( 1 , 129 );
09440            !196: INDS
09450            m1( 3 , 129 );
09460            !197: LDLA,LDLI,LDLB  group 3
09470            m2( 0 , 130 );
09480            !198: LDLR
09490            m2( 1 , 130 );
09500            !199: LDLS
09510            m2( 3 , 130 );
09520
09530            !200: LDOA,LDOI,LDOB  group 3
09540            m1( 0 , 130 );
09550            !201: LDOR
09560            m1( 1 , 130 );
09570            !202: LDOS
09580            m1( 3 , 130 );
09590
09600            !203: LODA,LODI,LODB  group 4
09610            m3( 0 , 130 );
09620            !204: LODR
09630            m3( 1 , 130 );
09640            !205: LODS
09650            m3( 3 , 130 );
09660
09670            !206: STOA,STOI,STOB  group 1
09680            m0( 0 , 131 );
09690            !207: STOR
09700            m0( 1 , 131 );
09710            !208: STOS
09720            m0( 3 , 131 );
```

```
09730
09740           !209: STLA,STLI,STLB  group 3
09750           m2( 0  , 132 );
09760           !210: STLR
09770           m2( 1  , 132 );
09780           !211: STLS
09790           m2( 3  , 132 );
09800
09810           !212: SROA,SROI,SROB  group 3
09820           m1( 0  , 132 );
09830           !213: SROR
09840           m1( 1  , 132 );
09850           !214: SROS
09860           m1( 3  , 132 );
09870
09880           !215: STRA,STRI,STRB  group 4
09890           m3( 0  , 132 );
09900           !216: STRR
09910           m3( 1  , 132 );
09920           !217: STRS
09930           m3( 3  , 132 );
09940         TES;
09950        exectwo();
09960       END; ! type2execzero
09970
09980      ROUTINE initinterpreter =
09990       BEGIN
10000        LOCAL trk,srfc,sectr;
10010        BIND
10020         input = 0  , inpadr = 10 ,
10030         output = 1 , outadr = 12 ,
10040         prd = 2  , prdadr = 14 ,
10050         prr = 3  , prradr = 16 ,
10060
10070        ppc = 0 ; psp = -1 ; mp = 0 ; np = maxstk; ! checkup np initial value
10080        interpret = true;
10090        ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Input");
10100        getfilename(trk,srfc,sectr);
10110        reset(input,pstk[inpadr],.trk,.srfc,.sectr);
10120        ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Prd");
10130        getfilename(trk,srfc,sectr);
10140        reset(prd  ,pstk[prdadr],.trk,.srfc,.sectr);
10150        ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Output");
10160        getfilename(trk,srfc,sectr);
10170        rewrite(output,pstk[outadr],.trk,.srfc,.sectr);
10180        ttcrlf(.ttyno); ttoutstr(.ttyno,UPLIT ASCIZ "Prr");
10190        getfilename(trk,srfc,sectr);
10200        rewrite(prr   ,pstk[prradr],.trk,.srfc,.sectr);
10210        IF .trace THEN
10220         BEGIN
10230          ttoutstr(.ttyno,UPLIT ASCIZ "   PSP STK[sp]                PPC  Opcode"); ttcrlf(.ttyno);
10240         END;
10250       END; ! initinterpreter
10260
```

```
10270       GLOBAL ROUTINE intr(trce,ttyout) =
10280        BEGIN
10290          ttcrlf(.ttyno);
10300          ttoutstr(.ttyno,version); ttcrlf(.ttyno);
10310          trace = .trce; outtty = .ttyout;
10320          initinterpreter();
10330          WHILE .interpret DO
10340           BEGIN
10350             op = locfetch();
10360             IF .trace THEN
10370              BEGIN
10380                ttputoct(.ttyno,.psp-1); putlch(.ttyno,space); ttputoct(.ttyno,.pstk[.psp-7]); putlch(.tt
10390                ttputoct(.ttyno,.pstk[.psp-5]); putlch(.ttyno,space); ttputoct(.ttyno,.pw); putlch(.ttyno
10400                 ttputoct(.ttyno,.cw); putlch(.ttyno,space);
10410                ttputoct(.ttyno,.ppc); putlch(.ttyno,space); ttputoct(.ttyno,.op); ttcrlf(.ttyno); NOP();
10420              END;
10430              IF NOT .op<t1bit> THEN execone() ! group 1
10440              ELSE IF .op<t2bit> THEN type2execzero()
10450                 ELSE type1()
10460           END;
10470           allclose();
10480           ttoutstr(.ttyno,mes1)
10490         END; ! intr
10500
10510        END
10520      ELUDOM
```

```
00010      %
00020      ************************************************************************
00030      **********                                                    **********
00040      **********  MODULE : SYSTEMLIBRARY                            **********
00050      **********  DATE : 1 JULY 1982                                **********
00060      **********  AUTHOR : S SRINIVAS RAGHURAM                      **********
00070      **********  ENTRIES: TTCRLF(TTYLIN); TTPUTOCT(TTYLIN,VAL);    **********
00080      **********           TTGETDEC(TTYLIN,VAL); INITTAPS();        **********
00090      **********           TTOUTSTR(TTYLIN,ADDR); SYSERR(N);        **********
00100      **********           GETFILENAME(TRK,SREC,SECTR);             **********
00110      **********           TYPE(TRK,SREC,SECTR);                    **********
00120      **********           COPY(ST,SC,SS,DT,DC,DS);                 **********
00130      **********           PSAVE(TRK,SREC,SECTR,MPC);               **********
00140      **********           PLOAD(TRK,SREC,SECTR);                   **********
00150      **********                                                    **********
00160      ************************************************************************%
00170      MODULE syslib(nolist)=
00180       BEGIN
00190         EXTERNAL getlch,putlch;
00200         EXTERNAL ttyno;
00210         EXTERNAL read,write,readln,writeln;
00220         EXTERNAL reset,rewrite,endofline,endoffile,fileclose;
00230         EXTERNAL open,close,getbyte,putbyte;
00240         EXTERNAL fetch,deposit;
00250
00260         BIND space = #40, nul = #0;
00270         BIND rd = 0, wr = 1, input = 0, output = 1;
00280         BIND false = 0, true = 1;
00290
00300         MACRO lsby = 0,8$, msby = 8,8$;
00310
00320
00330         GLOBAL ROUTINE ttcrlf(ttylin)=
00340           BEGIN
00350             putlch(.ttylin,#12);putlch(.ttylin,#15);
00360           END;
00370
00380         GLOBAL ROUTINE ttputoct(ttylin,val)=
00390          BEGIN
00400           LOCAL ch;
00410           ROUTINE octch(ttylin,ch)=
00420             putlch(.ttylin,#60+.ch);
00430           ch=.val<15,1>;octch(.ttylin,.ch);
00440           val=.val*2;
00450           INCR i FROM 1 TO 5 DO
00460            (ch=.val<13,3>;octch(.ttylin,.ch);
00470             val=.val*8);
00480          END;
00490
00500        GLOBAL ROUTINE ttgetdec(ttylin,val) =
00510                       ! val is a refrence parameter
00520        BEGIN
00530         LOCAL ch , i , n;
00540         n = 0; i = 0;
```

```
00550         getlch(.TTYLIN,cn);
00560         WHILE (.ch LEQ #71 AND .ch GEQ #60 AND .i LEQ 5 )DO
00570          BEGIN
00580           n = .n * 10 + .cn - #60;
00590           i = .i + 1;
00600           getlch(.ttylin,ch) ;
00610          END;
00620          IF .CH EQL #12 THEN putlch(.ttylin,#15)
00630            ELSE IF .ch EQL #15 THEN putlch(.ttylin,#12);
00640           .val = .n;
00650        END; ! ttgetdec
00660
00670        GLOBAL ROUTINE ttoutstr(ttylin,addr) =
00680                       ! addr is a reference parameter
00690         BEGIN
00700          WHILE .(.addr)<0,8> NEQ nul DO
00710           BEGIN
00720            putlch(.ttylin,.(.addr)<0,8>); addr = .addr + 1 ;
00730           END
00740        END;
00750
00760         ROUTINE EMT emttrap =
00770          BEGIN
00780           BIND progcnt = #177740, stkptr = #177742;
00790           EXTERNAL getcommand,savstkval;
00800            stkptr = .savstkval;
00810            progcnt = getcommand;
00820          END; ! emttrap
00830
00840        ROUTINE INTERRUPT oddadrtrap =
00850         BEGIN
00860          ttoutstr( .ttyno, UPLIT ASCIZ "ODD ADDRESS / ILL. INSTR. TRAP AT PC: ");
00870          ttputoct( .ttyno, .OLDPC );
00880          EMT(0)
00890         END; ! oddadrtrap
00900
00910        ROUTINE INTERRUPT stkovrfltrap =
00920         BEGIN
00930          ttoutstr( .ttyno, UPLIT ASCIZ "SYSTEM STACK OVERFLOW AT PC: ");
00940          ttputoct( .ttyno, .OLDPC );
00950          EMT(0)
00960         END; ! stkovrfltrap
00970
00980        ROUTINE INTERRUPT sometrap =
00990         BEGIN
01000          ttoutstr( .ttyno, UPLIT ASCIZ "SOME TRAP AT PC: ");
01010          ttputoct( .ttyno, .OLDPC );
01020          EMT(0)
01030         END; ! sometrap
01040
01050         GLOBAL  ROUTINE inittraps =
01060          BEGIN
01070           #20 = emttrap; #34 = oddadrtrap; #4 = stkovrfltrap;
01080           #22 = 0;       #36 = 0;          #6 = 0;
```

```
01090          #10 = sometrap; #14 = sometrap; #24 = sometrap;
01100          #12 = 0;        #16 = 0;        #26 = 0;
01110        #30 = sometrap; #14 = sometrap; #40 = sometrap;
01120        #32 = 0;        #16 = 0;        #42 = 0;
01130        #00 = sometrap; #02 = 0;
01140       END; ! inittraps
01150
01160      GLOBAL ROUTINE getfilename(trk,srfc,sectr)=
01170            ! trk,srfc,sectr are reference parameters
01180       BEGIN
01190        BIND
01200         mes1 = UPLIT ASCIZ " File name: ",
01210         mes2 = UPLIT ASCIZ "Track    No.: ",
01220         mes3 = UPLIT ASCIZ "Surface  No.: ",
01230         mes4 = UPLIT ASCIZ "Sector   No.: ",
01240         mes5 = UPLIT ASCIZ " ERROR! Retype";
01250
01260        ROUTINE bringnum(llim , hlim)=
01270         BEGIN
01280          LOCAL n,flag;
01290          flag = true;
01300          WHILE .flag DO
01310           BEGIN
01320            n = 0;
01330            ttgetdec(.ttyno,n);
01340            ! %debug%  putlch(.ttyno,space); ttputoct(.ttyno,.n);
01350            IF .n GEQ .llim AND .n LEQ .hlim THEN flag = false
01360             ELSE ttoutstr(.ttyno,mes5);
01370            ttcrlf(.ttyno);
01380           END;
01390         RETURN .n;
01400        END; ! bringnum
01410
01420        ttcrlf(.ttyno); ttoutstr(.ttyno,mes1); ttcrlf(.ttyno);
01430        ttoutstr(.ttyno,mes2); .trk=bringnum(100,200);
01440        ttoutstr(.ttyno,mes3); .srfc=bringnum(0,9);
01450        ttoutstr(.ttyno,mes4); .sectr=bringnum(1,10);
01460       END; ! getfilename
01470
01480      GLOBAL ROUTINE type(trk,srfc,sectr)=
01490       BEGIN
01500        LOCAL badr,ch,count;
01510        count = 0;
01520        reset(input,badr,.trk,.srfc,.sectr);
01530        UNTIL endoffile(input) DO
01540         BEGIN
01550          IF endofline(input) THEN
01560           BEGIN
01570            readln(input); ttcrlf(.ttyno); count = .count + 2
01580           END
01590          ELSE
01600           BEGIN
01610            read(input,ch); putlch(.ttyno,.ch); count = .count + 1
01620           END;
```

```
01630           END;
01640         fileclose(input);
01650         ttcrlf(.ttyno);
01660         ttoutstr(.ttyno,UPLIT ASCIZ "File size: ");
01670         ttputoct(.ttyno,.count + 1 ); putlch(.ttyno,"B");
01680         ttcrlf(.ttyno)
01690       END; ! type
01700
01710       GLOBAL ROUTINE copy(st,sc,ss,dt,dc,ds)=
01720             ! copy of ASCII files only!
01730       BEGIN
01740         LOCAL inpbadr,outbadr,ch;
01750         reset(input,inpbadr,.st,.sc,.ss);
01760         rewrite(output,outbadr,.dt,.dc,.ds);
01770         UNTIL endoffile(input) DO
01780           BEGIN
01790             IF endofline(input) THEN
01800               BEGIN
01810                 readln(input); writeln(output)
01820               END
01830             ELSE
01840               BEGIN
01850                 read(input,ch); write(output,.ch)
01860               END
01870           END;
01880         fileclose(input); fileclose(output)
01890       END; ! copy
01900
01910      GLOBAL ROUTINE syserr( n )=
01920       BEGIN
01930         IF .n LEQ 11 THEN
01940           ttoutstr(.ttyno,UPLIT ASCIZ "DISK ERROR: ")
01950         ELSE
01960           ttoutstr(.ttyno,UPLIT ASCIZ "Pascal File IO ERROR: ");
01970         ttputoct(.ttyno,.n);
01980         EMT(0);
01990       END; ! syserr
02000
02010     %
02020               DISK ERROR 1 : write attempted on a write protected disk
02030               DISK ERROR 2 : seek/search error ; non data transfer error
02040               DISK ERROR 3 : bus grant late
02050               DISK ERROR 4 : crcc error/read or write disable
02060               DISK ERROR 5 : synchronous error
02070               DISK ERROR 6 : drive unsafe
02080               DISK ERROR 7 : command check
02090               DISK ERROR10 : sector over run
02100               DISK ERROR11 : track over run
02110               DISK ERROR12 : non existant memory
02120               DISK ERROR13 : other errors
02130         PASCAL IO ERROR14 : attempt to get/read beyond EOF
02140         PASCAL IO ERROR15 : buffers exhausted cannot open file
02150         PASCAL IO ERROR16 : file has not been opened for write
02160         PASCAL IO ERROR17 : input data erroneous
```

```
02170         PASCAL ID ERROR20 : too many digits for an integer
02180       %
02190
02200        GLOBAL ROUTINE psave(trk,srfc,sectr,mpc) =
02210         BEGIN
02220          LOCAL fsp,flag,ch;
02230          open(.trk,.srfc,.sectr,wr,fsp,flag);
02240          IF .flag THEN syserr(13)
02250           ELSE
02260            BEGIN
02270             ttoutstr(.ttyno,UPLIT ASCIZ "Saving");
02280             ch = .mpc<lsby>; putbyte(.fsp,.ch);
02290             ch = .mpc<msby>; putbyte(.fsp,.ch);
02300             INCR ppc FROM 0 TO .mpc DO
02310              putbyte(.fsp,fetch(.ppc));
02320             close(.fsp); ttcrlf(.ttyno);
02330            END
02340          END; ! psave
02350
02360        GLOBAL ROUTINE pload(trk,srfc,sectr) =
02370         BEGIN
02380          LOCAL fsp,flag,ch,mpc;
02390          open(.trk,.srfc,.sectr,rd,fsp,flag);
02400          IF .flag THEN syserr(13)
02410           ELSE
02420            BEGIN
02430             ttoutstr(.ttyno,UPLIT ASCIZ "Loading");
02440             getbyte(.fsp,ch); mpc<lsby> = .ch;
02450             getbyte(.fsp,ch); mpc<msby> = .ch;
02460             INCR ppc FROM 0 TO .mpc DO
02470              BEGIN
02480              getbyte(.fsp,ch); deposit(.ppc,.ch)
02490             END;
02500            close(.fsp); ttcrlf(.ttyno)
02510           END
02520         END; ! pload
02530
02540
02550       END
02560     ELUDOM
```

```
00010      %
00020      ************************************************************************
00030      **********                                                    **********
00040      **********  MODULE : PASCALFILEOPERATORS                      **********
00050      **********  DATE : 1 JULY 1982                                **********
00060      **********  AUTHOR : S SRINIVAS RAGHUPAM                      **********
00070      **********  ENTRIES : GET(F); PUT(F); READ(F,CH); READLN(F);   **********
00080      **********            RESET(F,BADR,TRK,SREC,SECTR);            **********
00090      **********            RFWRITE(F,BADR,TRK,SREC,SECTR);          **********
00100      **********            WRITE(F,CH); WRTTELN(F); READINT(F,V);   **********
00110      **********            ENDOFFILE(F); ENDOFLINE(F);              **********
00120      **********            FILECLOSE(F);                            **********
00130      **********                                                     **********
00140      ************************************************************************%
00150      MODULE pascalfileoperators (nolist) =
00160       BEGIN
00170        BIND
00180         ! fields in the filetable
00190         fsp     =  0   ,   ! file pointer
00200         eoln    =  1   ,
00210         eof     =  2   ,
00220         auxch   =  3   ,   ! used for taking care of look ahead char for EOLN
00230         bufadr  =  4   ,   ! address of the one char file buffer on the stack
00240         ! file names code
00250         input = 0 , output = 1 , prd = 2 , prr = 3 ,
00260         ! constants
00270         lf = #12 , cr = #15 , sp = #40 , eofch = #32 , false = 0 , true = 1 ;
00280
00290        EXTERNAL getbyte , putbyte , open , close , syserr ;
00300
00310        STRUCTURE filetable [ fileno , field ] =
00320         [ fileno * field * bytes ]
00330         CASE .field OF
00340          SET
00350           ( .filetable + .fileno * bytes * field     )<0,8*bytes>; ! fsp
00360           ( .filetable + .fileno * bytes * field + 1 )<0,8*bytes>; ! eoln
00370           ( .filetable + .fileno * bytes * field + 2 )<0,8*bytes>; ! eof
00380           ( .filetable + .fileno * bytes * field + 3 )<0,8*bytes>; ! auxch
00390           ( .filetable + .fileno * bytes * field + 4 )<0,16>       ! bufadr
00400          TES;
00410
00420        BYTE OWN filetable ft[ 6 , 8 ];! 8 MUST BE 6 ,BUT PUT TO AVOID JSR PC,MUL
00430
00440        GLOBAL ROUTINE get( f ) =
00450         BEGIN
00460          IF  .ft[ .f , eof ] THEN syserr(12)
00470              ! attempt to get/read beyond EOF
00480          ELSE
00490           BEGIN
00500            IF .ft[ .f , auxch ] EQL lf THEN
00510              getbyte( .ft[ .f , fsp ] , .ft[ .f , bufadr ] )
00520               ! RISK : badr passed to reset must be a WORD address else ...
00530            ELSE
00540             BEGIN
```

```
00550             .ft[ .f , bufadr ] = .ft[ .f , auxch ];
00560              ft[ .f , auxch  ] = lf;
00570            END;
00580             IF ..ft[ .f , bufadr ] EQL eofch THEN
00590              BEGIN
00600                ft[ .f , eof ] = true; ft[ .f , eoln ] = true;
00610                RETURN;
00620              END
00630             ELSE
00640              IF ..ft[ .f , bufadr ] EQL cr THEN
00650               BEGIN
00660                LOCAL ch ;
00670                getbyte(.ft[ .f , fsp ] , ch);
00680                ft[ .f , auxch ] = .ch;
00690                IF .ch EQL lf THEN
00700                 BEGIN
00710                  .ft[ .f , bufadr ] = sp;
00720                   ft[ .f , eoln   ] = true;
00730                   RETURN;
00740                 END
00750               END;
00760               IF .ft[ .f , eoln ] THEN ft[ .f , eoln ] = false;
00770         END
00780       END; ! get
00790
00800      GLOBAL ROUTINE reset( f , badr , trk , srfc , sectr ) =
00810       BEGIN
00820        LOCAL tmp , flag;
00830        BIND rd = 0 ;
00840        ft[ .f , auxch ] = lf;
00850        ft[ .f , eoln  ] = false;
00860        ft[ .f , eof   ] = false;
00870        ft[ .f , bufadr] = .badr;
00880        open( .trk , .srfc , .sectr , rd , tmp , flag );
00890        ft[ .f , fsp ] = .tmp;
00900        IF NOT .flag THEN get(.f) ELSE syserr(13);
00910             ! buffers exhausted cannot open file
00920       END; ! reset
00930
00940      GLOBAL ROUTINE rewrite( f , badr , trk , srfc , sectr ) =
00950       BEGIN
00960        LOCAL tmp , flag;
00970        BIND wrt = 1;
00980        ft[ .f , auxch ] = lf;
00990        ft[ .f , eoln  ] = true;
01000        ft[ .f , eof   ] = true;
01010        ft[ .f , bufadr] = .badr;
01020        open( .trk , .srfc , .sectr , wrt , tmp , flag );
01030        ft[ .f , fsp ] = .tmp;
01040        IF .flag THEN syserr(13);
01050             ! buffers exhausted cannot open file
01060       END; ! rewrite
01070
01080      GLOBAL ROUTINE put(f)=
```

```
01090          BEGIN
01100           IF .ft[ .f , eof ] THEN
01110             putbyte( .ft[ .f , fsp ] , ..ft[ .f , bufadr ])
01120           ELSE syserr(14);
01130                ! file has not been opened for write
01140          END; ! put
01150
01160         GLOBAL ROUTINE readln(f)=
01170          BEGIN
01180           WHILE NOT .ft[ .f , eoln ] DO get(.f);
01190           IF NOT .ft[ .f , eof ] THEN get(.f);
01200          END; ! readln
01210
01220         GLOBAL ROUTINE writeln(f)=
01230          BEGIN
01240           .ft[ .f , bufadr ] = cr ; put(.f);
01250           .ft[ .f , bufadr ] = lf ; put(.f);
01260          END; ! writeln
01270
01280         GLOBAL ROUTINE read(f,ch)=
01290          ! ch is a reference parameter
01300          BEGIN
01310           IF NOT .ft[ .f , eof ] THEN
01320            BEGIN
01330             .ch = ..ft[ .f , bufadr ];
01340                   get(.f);
01350            END
01360           ELSE syserr(12)
01370                ! attempt to get/read beyond EOF
01380          END; ! read
01390
01400         GLOBAL ROUTINE write(f,ch)=
01410          BEGIN
01420           .ft[ .f , bufadr ] = .ch;
01430           put(.f);
01440          END; ! write
01450
01460         GLOBAL  ROUTINE readint( f , v ) =
01470                ! v is a reference parameter
01480          BEGIN
01490           LOCAL
01500                k   ,   ! digit count
01510                n   ,   ! current value of integer
01520                s   ;   ! sign information
01530           BIND space = #40, plus = #53, minus = #55, zero = #60, nine = #71;
01540           BIND nmax = 5;  ! maximum no. of digits in an integer
01550           BIND buf = .ft[ .f , bufadr ];
01560
01570           k = 0; n = 0; s = 0;
01580           WHILE .buf EQL space DO  get( .f );
01590           IF .buf EQL plus THEN get( .f )
01600           ELSE IF .buf EQL minus THEN
01610            BEGIN
01620             get( .f ); s = 1;
```

```
01630           END;
01640         WHILE .buf GEQ zero AND .buf LEQ nine DO
01650           BEGIN
01660           n = 8 * .n + 2 * .n + .buf - zero;
01670           k = .k + 1;
01680           get( .f );
01690           END;
01700         IF .k EQL 0 THEN syserr(15); ! input data erroneous
01710         IF .k GTR nmax THEN syserr(16); ! too many digits for an integer
01720           ! NOTE: for integers between +/-32768 to +/-99999 results are all balls*
01730         IF .s THEN n = -.n;
01740         .v = .n;
01750        END; ! readint
01760
01770       GLOBAL ROUTINE endoffile(f) = .ft[ .f , eof ];
01780
01790       GLOBAL ROUTINE endofline(f) = .ft[ .f , eoln];
01800
01810       GLOBAL ROUTINE fileclose(f) = close( .ft[ .f , fsp ]);
01820
01830       %debug
01840       GLOBAL ROUTINE dmpfiltab(f)=
01850         BEGIN
01860         EXTERNAL ttputoct,ttcrlf,putlch;
01870         EXTERNAL ttyno;
01880         MACRO o1(what)=ttputoct(.ttyno,.ft[.f,what]) $,
01890                p1=putlch(.ttyno,sp) $;
01900          o1(fsp);p1;o1(eoln);p1;o1(eof);p1;o1(auxch);p1;
01910          o1(bufadr);ttcrlf(.ttyno);
01920         END; ! dmpfiltab
01930       debug%
01940
01950
01960     end
01970     eludom
```

```
00010      %
00020      ************************************************************************
00030      **********                                                    **********
00040      **********   MODULE : PSEUDOFILER                             **********
00050      **********   DATE : 1 JULY 1982                               **********
00060      **********   AUTHOR : S SRINIVAS RAGHURAM                     **********
00070      **********   ENTRIES: GETBYTE(FSP,CH); PUTBYTE(FSP,CH);       **********
00080      **********            OPEN(TRACK,SURFACE,SECTOR,FOR,FSP,FLAG);**********
00090      **********            CLOSE(FSP); INITBUFS(); ALLCLOSE();      **********
00100      **********                                                    **********
00110      ************************************************************************%
00120      MODULE pseudofiler( nolist )=
00130       BEGIN
00140        MACRO
00150         purpose = 8,8 $ ,
00160         bufptr  = 0,8 $ ,
00170         trk     = 8,8 $ ,
00180         srfc    = 4,4 $ ,
00190         str     = 0,4 $ ;
00200
00210        BIND
00220         ! fields of openfilestable
00230         baseadr = 0 ,
00240         curadr  = 1 ,
00250         purbuf  = 2 ,
00260         nofs    = 6 ,  ! number of open files (max)
00270         freeslot= 0 ,
00280         rd      = 0 ,
00290         wr      = 1 ,
00300         undef   = 2 ,
00310         false   = 0 ,
00320         true    = 1 ,
00330         eofch   =#32,
00340         bufsize =#200 ; ! buffer size = sector size ;(words)
00350
00360        STRUCTURE openfilestab[ fp , i ] =
00370         CASE .i OF
00380          SET
00390           ( .openfilestab + .fp * 8     );  ! baseadr
00400           ( .openfilestab + .fp * 8 + 2 );  ! curadr
00410           ( .openfilestab + .fp * 8 + 4 );  ! purbuf
00420          TES;
00430
00440        STRUCTURE buftab[ bufno , index ] =
00450         [ bufno * index * bytes ]
00460          ( .buftab + .bufno * 2 * bufsize * bytes + .index * bytes )<0,8*bytes>;
00470
00480          OWN openfilestab opnf[ nofs * 4 ]; ! 4 ==> 3 , TO GET OVER MUL IN CODE
00490
00500        BYTE  OWN buftab bfr[ nofs , 2 * bufsize ];
00510
00520        EXTERNAL getblk , putblk ;
00530
00540        %debug
```

```
00550       EXTERNAL ttcrlf,ttputoct;
00560       EXTERNAL ttyno;
00570     debug%
00580
00590       ROUTINE inccuradr( fsp ) =
00600        BEGIN
00610         BIND k = opnf[ .fsp , curadr ];
00620         k<str> = .k<str> + 1;
00630         IF .k<str> GTR 10 THEN
00640          BEGIN
00650           k<str> = 1;
00660           k<srfc> = .k<srfc> + 1;
00670           IF .k<srfc> GTR 9 THEN
00680            BEGIN
00690             k<srfc> = 0;
00700             k<trk> = .k<trk> + 1
00710            END
00720          END
00730        END; ! inccuradr
00740
00750       GLOBAL ROUTINE getbyte( fsp , ch ) =
00760        ! ch is a reference parameter
00770        BEGIN
00780         BIND k = opnf[ .fsp , purbuf ];
00790         IF .k<purpose> EQL rd THEN
00800          BEGIN
00810           k<bufptr> = .k<bufptr> + 1;
00820           IF .k<bufptr> EQL 0 THEN
00830            BEGIN
00840             BIND j = opnf[ .fsp , curadr ];
00850             getblk( .j<trk> , .j<srfc> , .j<str> , bufsize , bfr[ .fsp , 0 ]);
00860             inccuradr( .fsp )
00870            END;
00880           .ch = .bfr[.fsp, .k<bufptr> ]
00890          END
00900        END; ! getbyte
00910
00920
00930       GLOBAL ROUTINE putbyte( fsp , ch ) =
00940        BEGIN
00950         BIND k = opnf[ .fsp , purbuf ];
00960         IF .k<purpose> EQL wr THEN
00970          BEGIN
00980           bfr[.fsp, .k<bufptr> ] = .ch;
00990           k<oufptr> = .k<bufptr> + 1;
01000           IF .k<bufptr> EQL 0 THEN
01010            BEGIN
01020             BIND j = opnf[ .fsp , curadr ];
01030             putblk( .j<trk> , .j<srfc> , .j<str> , bufsize , bfr[ .fsp , 0 ]);
01040             inccuradr( .fsp )
01050            END;
01060          END
01070        END; ! outbyte
01080
```

```
01090       GLOBAL ROUTINE open( track , surface , sector , for , fsp , flag ) =
01100        ! fsp and flag are reference parameters ; flag = true ==> buffers
01110        !  are not available
01120
01130        BEGIN
01140         LOCAL ptr , adr ;
01150         adr<trk> = .track;  adr<srfc> = .surface;
01160         adr<str> = .sector; ptr = nofs;
01170         INCR k FROM 0 TO nofs-1 DO
01180          BEGIN
01190           BIND j = .opnf[ .k , baseadr ];
01200           IF j EQL .adr THEN
01210            BEGIN
01220             ptr = .k; EXITLOOP
01230            END;
01240           IF j EQL freeslot THEN ptr = .k
01250          END;
01260          IF .ptr EQL nofs THEN
01270           BEGIN
01280            .flag = true; RETURN
01290           END
01300          ELSE
01310           BEGIN
01320            .flag = false; .fsp = .ptr;
01330            opnf[ .ptr , baseadr ] = .adr;
01340            opnf[ .ptr , curadr  ] = .adr;
01350            opnf[ .ptr , purbuf  ]<purpose> = .for;
01360            opnf[ .ptr , purbuf  ]<buffer > = (IF .for EQL rd THEN 255 ELSE 0);
01370           END
01380        END; ! open
01390
01400       GLOBAL ROUTINE close(fsp) =
01410        BEGIN
01420         IF .opnf[ .fsp , purbuf ]<purpose> EQL wr THEN
01430          BEGIN
01440           BIND k = opnf[ .fsp , curadr ];
01450           putbyte( .fsp , eofch );
01460           putblk( .k<trk> , .k<srfc> , .k<str> , bufsize , bfr[ .fsp , 0 ])
01470          END;
01480         opnf[ .fsp , baseadr ] = freeslot
01490        END; ! close
01500
01510       GLOBAL ROUTINE initbufs =
01520        BEGIN
01530         INCR k FROM 0 TO nofs-1 DO
01540          BEGIN
01550           opnf[ .k , baseadr ] = freeslot;
01560           opnf[ .k , purbuf  ]<purpose> = undef
01570          END;
01580         END; ! initbufs
01590
01600       GLOBAL ROUTINE allclose =
01610        INCR k FROM 0 TO nofs-1 DO
01620         IF .opnf[ .k , baseadr ] NEQ freeslot THEN close( .k ); ! allclose
```

```
01630
01640       %debug
01650         GLOBAL ROUTINE dmpbuf(fsp,ll,hl)=
01660          INCR i FROM .ll TO .hl DO
01670           ( ttputoct(.ttyno,.bfrl.fsp,.i)); ttcrlf(.ttyno));
01680
01690         GLOBAL ROUTINE dmptab(fsp)=
01700          BEGIN
01710           ttputoct(.ttyno,.opnfl.fsp,baseadr); ttputoct(.ttyno,.opnfl.fsp,curadr);
01720           ttcrlf(.ttyno);
01730           ttputoct(.ttyno,.opnfl.fsp,purpose)<purpose>);
01740           ttputoct(.ttyno,.opnfl.fsp,curbuf)<bufptr>); ttcrlf(.ttyno);
01750          END; ! dmptab
01760       ndebug%
01770
01780        END
01790       ELUDOM
01800
```

```
00010      %
00020      ************************************************************************
00030      **********                                                      **********
00040      **********  MODULE : DISKIO                                     **********
00050      **********  DATE : 1 JULY 1982                                  **********
00060      **********  AUTHOR : S SRINIVAS RAGHURAM                        **********
00070      **********  ENTRIES: PUTBLK(PTRK,PSRFC,PSTR,PWRDCONT,PADDR);    **********
00080      **********           GETBLK(PTRK,PSRFC,PSTR,PWRDCONT,PADDR);    **********
00090      **********           DISKINIT();                                **********
00100      **********                                                      **********
00110      ************************************************************************%
00120      MODULE diskio (NOLIST) =
00130      BEGIN
00140        MACRO
00150          ! command register dcr
00160              seek       = 0,1 $ ,        ! seek bit
00170              search     = 1,1 $ ,        ! search bit
00180              read       = 2,1 $ ,        ! read bit
00190              write      = 3,1 $ ,        ! write bit
00200              restore    = 4,1 $ ,        ! restore bit
00210              writechk   = 5,1 $ ,        ! write check bit
00220              enabl      = 6,1 $ ,        ! interrupt enable bit
00230              unit       =10,1 $ ,        ! unit no
00240              port       =11,2 $ ,        ! 2 bit port address
00250              varb       =13,1 $ ,        ! variable / fixed format
00260
00270          ! drive status register dsr
00280              sercom     = 0,1 $ ,        ! search complete
00290              sekcom     = 1,1 $ ,        ! seek complete
00300              oprejt     = 4,1 $ ,        ! operation rejected
00310              donlin     = 5,1 $ ,        ! drive online
00320              ready      = 6,1 $ ,        ! ready to seek,read or write
00330              writeprot  =15,1 $ ,        ! port write protected
00340
00350           ! controller status register csr
00360              busy       = 0,1 $ ,        ! data control busy
00370              bsgrtl     = 1,1 $ ,        ! bus grant late
00380              crcerr     = 2,1 $ ,        ! crcc error
00390              syncerr    = 3,1 $ ,        ! synchronous error
00400              rwdsble    = 6,1 $ ,        ! r/w disable
00410              conready   = 7,1 $ ,        ! controller ready
00420              dunsafe    = 8,1 $ ,        ! drive unsafe
00430              cmmdchk    = 9,1 $ ,        ! command check
00440              sovrun     =10,1 $ ,        ! sector over run
00450              tovrun     =11,1 $ ,        ! track  over run
00460              nonexmm    =12,1 $ ,        ! non existant memory
00470              wrdctofl   =13,1 $ ,        ! word count overflow
00480              done       =14,1 $ ,        ! done flag
00490              error      =15,1 $ ,        ! error flag
00500
00510              sectr      = 0,5 $ ,        ! sector address field in str
00520              srface     = 5,4 $ ,        ! surface address field in str
00530              bits4      = 0,4 $ ,        ! first 4 bits
00540              bits5      = 0,5 $ ;        ! first 5 bits
```

```
00550
00560       BIND
00570             str      = #177444    ,     ! disk sector and surface address  register
00580             dsr      = #177450    ,     ! drive status register
00590             trk      = #177452    ,     ! disk track address register
00600             dcr      = #177454    ,     ! command register
00610             csr      = #177456    ,     ! controller status register
00620             adar     = #177460    ,     ! current address register
00630             wrd      = #177462    ,     ! word count register
00640             dtvt     = #200       ,     ! data transfer interrupt vector
00650             ndtvt    = #204       ,     ! non data transfer interrupt vector
00660             status   = #104000    ,     ! in command reg ,sense=1; port=01
00670             cmmdinit= #004100    ,     ! port=01; int enable=1
00680             wrtechk  = #004140    ,     ! port=01; int enb=1; writechk =1
00690             restre   = #004020    ,     ! port=01; restore=1
00700             secsize  = #200       ;     ! sector size is 128 words
00710                      % result values returned by putblk/getblk
00720                              to the monitor via EMT
00730                              result = 0 : operation completed/word count overflow
00740                              result = 1 : write attempted on a write protected disk
00750                              result = 2 : seek/search error ; non data transfer error
00760                              result = 3 : bus grant late
00770                              result = 4 : crcc error/read or write disable
00780                              result = 5 : synchronous error
00790                              result = 6 : drive unsafe
00800                              result = 7 : command check
00810                              result = 8 : sector over run
00820                              result = 9 : track over run
00830                              result =10 : non existant memory
00840                              result =11 : other errors
00850                                                                          %
00860       OWN
00870             cmmdword  , result , attempts ;
00880
00890       EXTERNAL syserr;
00900
00910       ROUTINE seeksearchdo ( ptrk , psrfc , pstr ) =
00920         BEGIN
00930             str < sectr > = .pstr < bits5 > ;
00940             IF ( .ptrk  NEQ  .trk ) or ( .psrfc  NEQ  .str < srface > ) THEN
00950               BEGIN
00960                 trk = .ptrk;
00970                 str < srface > = .psrfc < bits4 >;
00980                 cmmdword < seek > = 1
00990               END;
01000             cmmdword < search > = 1;
01010             dcr = .cmmdword          ! load and go to disk controller
01020         END;    ! seeksearchdo
01030
01040       ROUTINE commonchores ( pwrdcont , paddr ) =
01050         BEGIN
01060           UNTIL .csr < conready > DO ;    ! busy wait for controller to getready
01070           DO dcr = status UNTIL .dsr < ready > ;  ! busy wait for drive to getready
01080           attempts = 3;  !  3 attempts are made to read / write
```

```
01090           cmmdword = cmmdinit ;
01100           wrd = .pwrdcont;
01110           addr = .paddr;
01120           cmmdword < varb > = ( IF .pwrdcont  GTR  secsize  THEN  1  ELSE  0 )
01130         END;    !  commonchores
01140
01150       GLOBAL ROUTINE putblk ( ptrk , psrfc , pstr , pwrdcont , paddr  ) =
01160         BEGIN
01170           commonchores ( .pwrdcont , .paddr );
01180           cmmdword < write > = 1;
01190           dcr = status;
01200           IF .dsr < writeprot > THEN result = 1
01210           ELSE
01220               DO
01230                BEGIN
01240                 seeksearchdo ( .ptrk , .psrfc , .pstr ) ;
01250                 WAIT();
01260                END
01270               UNTIL ( .attempts LEQ 0 ) or ( .result neq  4 );
01280               IF ( .result NEQ 0 ) THEN syserr(.result);
01290         END;      ! putblk
01300
01310       GLOBAL ROUTINE getblk ( ptrk , psrfc , pstr , pwrdcont , paddr  ) =
01320         BEGIN
01330           commonchores ( .pwrdcont , .paddr );
01340           cmmdword < read > = 1;
01350           DO
01360            BEGIN
01370              seeksearchdo ( .ptrk , .psrfc , .pstr ) ;
01380              WAIT();
01390            END
01400           UNTIL ( .attempts LEQ 0 ) or ( .result NEQ 4 ) ;
01410           IF ( .result NEQ 0  ) THEN syserr(.result);
01420         END;      ! getblk
01430
01440       ROUTINE INTERRUPT nondata =
01450         BEGIN
01460           LOCAL  temp ;
01470           temp = .csr ;
01480           IF .temp < error > THEN  result = 2
01490                              ELSE result = 0
01500         END;  ! nondata
01510
01520       ROUTINE INTERRUPT data =
01530         BEGIN
01540          LOCAL  temp ;
01550           temp = .csr ;
01560           result = ( IF .temp < error > THEN
01570                        IF .temp < bsgrtl > THEN 3
01580                      ELSE IF .temp < crcerr > THEN 4
01590                      ELSE IF .temp < syncerr > THEN 5
01600                      ELSE IF .temp < rwdsble > THEN 4
01610                      ELSE IF .temp < dunsafe > THEN 6
01620                      ELSE IF .temp < cmmdchk > THEN 7
```

```
01630                         ELSE IF .temp < sovrun  > THEN 8
01640                         ELSE IF .temp < tovrun  > THEN 9
01650                         ELSE IF .temp < nonexmm > THEN 10
01660                         ELSE IF .temp < wrdctofl > THEN 0
01670                         ELSE 11
01680                       ELSE 0 );
01690           attempts = .attempts - 1;
01700         END;    ! data
01710
01720       GLOBAL ROUTINE diskinit =
01730         BEGIN
01740           dtvt = data ;  ! put address of data interrupt service routine
01750           ( dtvt + 2 ) = #240 ;  ! ps ; priority 5, gpr 1
01760           ndtvt = nondata ;
01770           ( ndtvt + 2 ) = #240;
01780           DO dcr = status UNTIL .dsr < donlin > ;  ! later put a message to operator
01790           trk = 0; str = 0;
01800           dcr = restre
01810         END;    ! diskinit
01820
01830
01840       END ELUDOM
```

```
00010      %
00020      ************************************************************************
00030      **********                                                    **********
00040      **********  MODULE : ABORTMAP                                 **********
00050      **********  DATE : 1 JULY 1982                                **********
00060      **********  AUTHOR : S SRINIVAS RAGHURAM                      **********
00070      **********  INTERRUPT SERVICE ROUTINE FOR MAPABORT            **********
00080      **********   RESTORES REGISTERS AND CALLS SWAPPOLICY           **********
00090      **********                                                    **********
00100      ************************************************************************%
00110      ;    GLOBAL ROUTINE INTERRUPT abortmap =
00120      ;         ! PRIORITY LEVEL = 4;
00130      ;         ! REGISTER SET USED IS 1
00140      ;         ! SUPERVISOR MODE
00150      ;     BEGIN
00160      ;      %.The MAPABORT interrupt service routine uses register set 1 ;
00170      ;        Any program segment using register set 1 should not cause a
00180      ;        Map Abort condition . If caused results are unpredictable
00190      ;%
00200      ;  LABEL  12 , 15 ;
00210      ;       INLINE("MOV     (SP)+,@#PCOLD  ");
00220      ;       INLINE("MOV     (SP)+,@#PSOLD  ");
00230      ;       ! now all registers except PC are in their old state
00240      ;       ! RESTORE REGISTERS
00250      ;       ! note again : ONLY REGISTER SET 0 CONSIDERED FOR RESTORATION
00260      ;       %debug% dmpregs();
00270      ;HALT();
00280      ;
00290      ; 12: BEGIN
00300      ;       t1 =  2 * .mapsr1<8,3> + rsba;
00310      ;           ! address calculation of register modified during the execution
00320      ;           ! of the instruction that caused MAP ABORT
00330      ;       IF .t1 EQL rsba THEN LEAVE 12;
00340      ;       t2 = (IF .mapsr1 LSS 0 THEN -1 ELSE 0);
00350      ;           ! the 3 bit count given by mapsr1 is in 2's complement form
00360      ;       t2<0,2> = .mapsr1<13,2>; ! load count
00370      ;       .t1 = ..t1 - .t2; ! restore register contents
00380      ;       END;
00390      ;
00400      ;           ! repeat for the next register
00410      ; 15: BEGIN
00420      ;       t1 =  2 * .mapsr1<0,3> + rsba;
00430      ;       IF .t1 EQL rsba THEN LEAVE 15;
00440      ;       t2 = (IF .mapsr1<0,8> LSS 0 THEN -1 ELSE 0);
00450      ;       t2<0,2> = .mapsr1<5,2>;
00460      ;       .t1 = ..t1 - .t2;
00470      ;       END;
00480      ;        pcold = .mapsr2;
00490      ;
00500      ;        INLINE(" MOV    @#PSOLD,-(SP) ");
00510      ;        INLINE(" MOV    @#PCOLD,-(SP) ");
00520      ;        %debug% dmpregs();
00530      ;HALT();
00540      ;
```

```
00550       ;       swappolicy();
00560       ;       %debug% dmpcoretable();
00570       ;HALT();
00580       ;       END; ! abortmap
00590
00600       R$0=%0
00610       R$1=%1
00620       R$2=%2
00630       R$3=%3
00640       R$4=%4
00650       R$5=%5
00660       SP=%6
00670       PC=%7
00680
00690
00700
00710
00720       ABORTMAP:
00730               MOV     (SP)+,@#PCOLD
00740               MOV     (SP)+,@#PSOLD
00750               ;JSR    PC,DMPREGS          ;DEBUG
00760               ;HALT                       ;DEBUG
00770               MOV     #T1,R$0
00780               MOV     @#174202,R$1
00790               SWAB    R$1
00800               BIC     #-7,R$1
00810               ASL     R$1
00820               ADD     #177740,R$1
00830               MOV     R$1,@R$0
00840               CMP     R$1,#-40
00850               BEQ     L2
00860               TST     @#174202
00870               BGE     L3
00880               MOV     #-1,T2
00890               BR      L4
00900       L3:     CLR     T2
00910       L4:     MOV     @#174202,R$2
00920               ROL     R$2
00930               ROL     R$2
00940               ROL     R$2
00950               ROL     R$2
00960               BIC     #-4,R$2
00970               BICB    #3,@#T2
00980               BISB    R$2,@#T2
00990               SUB     @#T2,@R$1
01000       L2:     MOVB    @#174202,R$1
01010               BIC     #-7,R$1
01020               ASL     R$1
01030               ADD     #177740,R$1
01040               MOV     R$1,@R$0
01050               CMP     R$1,#-40
01060               BEQ     L5
01070               TSTB    @#174202
01080               BHIS    L6
```

```
01090                   MOV     #-1,T2
01100                   BR      L7
01110           L6:     CLR     T2
01120           L7:     MOVB    @#174202,R$2
01130                   ASL     R$2
01140                   ASL     R$2
01150                   ASL     R$2
01160                   SWAB    R$2
01170                   BIC     #-4,R$2
01180                   BICB    #3,@#T2
01190                   BISB    R$2,@#T2
01200                   SUB     @#T2,@R$1
01210           L5:     MOV     @#174204,PCOLD
01220                   MOV   @#PSOLD,-(SP)
01230                   MOV   @#PCOLD,-(SP)
01240                   ;JSR    PC,DMPREGS              ;DEBUG
01250            ;      HALT                            ;DEBUG
01260                   JSR     PC,SWAPPOLICY
01270                   JSR     PC,DMPCORETAB           ;DEBUG
01280                   HALT                            ;DEBUG
01290                   RTI
01300
01310
01320                   .CSECT ABRT.O
01330                   T1:  .=.+2
01340                   T2:  .=.+2
01350                   PCOLD:  .=.+2
01360                   PSOLD:  .=.+2
01370
01380                   .GLOBL  ABORTMAP
01390                   .GLOBL  SWAPPOLICY
01400                   .GLOBL  DMPREGS                 ;DEBUG
01410                   .GLOBL  DMPCORETAB              ;DEBUG
01420
01430                   .END
```

```
00010      %
00020      ************************************************************************
00030      **********                                                    **********
00040      **********  MODULE : MEMORYMANAGER                            **********
00050      **********  DATE : 1 JULY 1982                                **********
00060      **********  AUTHOR : S SRINIVAS RAGHURAM                      **********
00070      **********  ENTRIES: SWAPPOLICY(); INITMAP();                 **********
00080      **********                                                    **********
00090      ************************************************************************%
00100      MODULE memorymanager (  list    ) =
00110
00120       BEGIN
00130        MACRO
00140              ! subfields in segdes field of coretable
00150         segtyp    =  3,2 $ ,
00160         segno     =  0,3 $ ,
00170
00180               ! fields in segmenttable descriptor register
00190         w         =  6,1 $ ;  ! segment written into
00200
00210        MACRO
00220         incparcelholder =
00230           parcelholder = .parcelholder + 1;
00240           IF .parcelholder GTR coretabmax THEN
00250            parcelholder = 0  $;
00260
00270       MACRO
00280         passparcel =
00290          DO
00300           ( incparcelholder)
00310          WHILE .ctl.parcelholder,resident] $;
00320
00330        BIND
00340          !  of hwsegment table
00350         mdr          = 0 ,     ! map descriptor register
00360         mar          = 1 ,     ! map address register
00370
00380          !  of core table
00390         segdes       = 0 ,     ! segment descriptor
00400         resident     = 1 ,     ! permanently resident segment
00410
00420          ! registers / addresses
00430         mapsr0       = #174200 ,      ! map status register 0
00440         mapsr1       = #174202 ,      ! map status register 1
00450         mapsr2       = #174204 ,      ! map status register 2
00460         rsba         = #177740 ,      ! register set base address
00470         mapabt       = #54     ,      ! MAPABORT interrupt vector
00480         maptrp       = #44     ,      ! MAPTRAP  interrupt vector
00490         outcore      = #077400 ,
00500
00510          ! for swapping
00520         onek         = 1024    ,      ! page size in words
00530         twok         = 2048    ,      ! page size in bytes
00540         segsize      = 8192    ,      ! segment size in bytes
```

```
00550
00560          ! constants
00570         du            = 0         ,      ! user data space
00580         ds            = 1         ,      ! supervisor data space
00590         iu            = 2         ,      ! user instruction space
00600         is            = 3         ,      ! supervisor instruction space
00610         free          =#40          ,    ! free space
00620         coretabmax    = 7         ,      !
00630         segtabmax     =31         ,      !
00640         false         = 0         ,      !
00650         true          = 1         ;      !
00660
00670        STRUCTURE nwsegmenttable [ i1 , i2 , j ] =
00680          CASE .j OF
00690           SET
00700            (.nwsegmenttable + .i1 * 32 + 2 * .i2);
00710            (.nwsegmenttable + .i1 * 32 + 2 * .i2 + 16)
00720           TES;
00730
00740        STRUCTURE coretable [ i , j ] =
00750         CASE  .j  OF
00760          SET
00770           ( .coretable + .i * 4     );
00780           ( .coretable + .i * 4 + 2 );
00790          TES;
00800
00810         STRUCTURE swsegmenttable [ i ] =
00820          ( .swsegmenttable + 2 * .i );
00830
00840         EXTERNAL putblk , getblk ;
00850         %debug% external putlch,ttputoct,ttcrlf,ttyno;
00860
00870          OWN coretable ct[ (coretabmax + 1) * 2 ];
00880
00890         BIND nwsegmenttable st = #174000;
00900
00910         OWN swsegmenttable swst [ segtabmax + 1 ];
00920
00930         EXTERNAL abortmap;
00940
00950         OWN parcelholder;
00960
00970          ROUTINE loadseg ( virtsegno , physegno ) =
00980          BEGIN
00990           LOCAL strtadr;
01000           strtadr = .physegno * segsize ;
01010           st[ .virtsegno<segtyp> , .virtsegno<segno> , mar ] = .strtadr DIV 64 ;
01020           st[ .virtsegno<segtyp> , .virtsegno<segno> , mdr ] = .swst[ .virtsegno ] ;
01030           ct[ .physegno , segdes ] = .virtsegno;
01040           ct[ .physegno , resident  ] = false;
01050           INCR srfc FROM 0 TO 3 DO
01060            BEGIN
01070             getblk( .virtsegno , .srfc , 2 , onek , .strtadr );
01080             strtadr = .strtadr + twok
```

```
01090           END;
01100         END;  ! loadseg
01110
01120         ROUTINE storeseg ( virtsegno , physegno ) =
01130         BEGIN
01140          LOCAL strtadr;
01150          strtadr = .physegno * segsize;
01160          st[ .virtsegno<segtyp> , .virtsegno<segno> ,mdr ] = 0;
01170          ct[ .physegno , segdes ] = free;
01180          INCR srfc FROM 0 TO 3 DO
01190           BEGIN
01200            putblk( .virtsegno , .srfc , 2 , onek , .strtadr );
01210            strtadr = .strtadr + twok
01220           END;
01230         END; ! storeseg
01240
01250
01260       GLOBAL ROUTINE   swappolicy =
01270        BEGIN
01280          LOCAL k; ! find the segment that caused the fault
01290          k = 0;
01300          k<3,1> = NOT .mapsr0<6,1>; ! U/S mode
01310          k<4,1> = NOT .mapsr0<5,1>; ! D/I space
01320          k<0,3> = .mapsr0<2,3>; ! segment number
01330         IF .ct[ .parcelholder , segdes ] EQL .k THEN
01340          BEGIN
01350           LOCAL k1;
01360           st[ .k<segtyp> ,.k<segno> ,mdr] = .swst[ .k ];
01370           passparcel;
01380           k1 = .ct[ .parcelholder , segdes ];
01390           st[ .k1<segtyp> , .k1<segno> ,mdr ] = outcore
01400          END
01410         ELSE
01420          BEGIN
01430           BIND k1 = ct[ .parcelholder , segdes ];
01440           IF .st[ .k1<segtyp>, .k1<segno> , mdr ]<w> THEN
01450            storeseg(.k1,.parcelholder)
01460           ELSE
01470            st[ .k1<segtyp> , .k1<segno> , mdr ] = 0;
01480           loadseg( .k, .parcelholder );
01490           passparcel;
01500          END;
01510         mapsr0<12,4> = 0 ; ! reset bits to resume MAP functioning from next instruction
01520        END; ! swappolicy
01530       %debug%
01540        GLOBAL ROUTINE dmpcoretable=
01550         BEGIN
01560          LOCAL n;
01570          INCR i FROM 0 TO coretabmax DO
01580           BEGIN
01590            n = .ct[ .i , segdes ]<segtyp>;
01600            ttputoct(.ttyno,.n); putch(.ttyno,#40);
01610            n = .ct[ .i , segdes ]<segno>;
01620            ttputoct(.ttyno,.n); putch(.ttyno,#40);
```

```
01630             n = .ctl[.i, resident];
01640             ttputoct(.ttyno,.n); ttcrlf(.ttyno)
01650            END;
01660           ttputoct(.ttyno,.parcelholder)
01670          END; ! dmpcoretable
01680
01690         GLOBAL ROUTINE dmpregs=
01700          BEGIN
01710           BIND strtadr = #177744;
01720           INCR i FROM 0 TO 6 BY 2 DO
01730            BEGIN
01740             ttputoct(.ttyno,.(strtadr+.i)); putch(.ttyno,#40)
01750            END;
01760           ttcrlf(.ttyno)
01770          END; ! dmpregs
01780       %debug%
01790
01800
01810         GLOBAL ROUTINE initmap =
01820          BEGIN
01830           INCR i FROM 0 TO segtabmax DO
01840            BEGIN
01850             swstl[.i] = #077407; !maplf=#177 ; ACF=#7
01860             st[.i<segtyp>, .i<segno>, mdr] = #077400; !maplf=#177 ; ACF=#0
01870            END;
01880           INCR i FROM 0 TO coretabmax DO
01890            BEGIN
01900            ctl[.i, segdes] = free; !free
01910            ctl[.i, resident] = false;
01920            END;
01930            st[is, 0, mar] = 0; !initialize seg0 to present code
01940            st[is, 7, mar] = #7600; ! registers
01950            st[is, 0, mdr] = #077407; ! maplf=#177, acf = #7
01960            st[is, 7, mdr] = #077407;
01970            st[ds, 7, mar] = #7600; st[ds, 7, mdr] = #077407;
01980            st[iu, 7, mar] = #7600; st[iu, 7, mdr] = #077407;
01990            st[du, 7, mar] = #7600; st[du, 7, mdr] = #077407;
02000            ctl[0,segdes] = #30; !is,seg0
02010            ctl[0, resident] = true;
02020            ctl[7,segdes] = #37; !is,seg7
02030            ctl[7, resident] = true;
02040            ctl[6, resident] = true;  !debug
02050            ctl[5, resident] = true;  !debug
02060            ctl[4, resident] = true;  !debug
02070            ctl[3, resident] = true;  !debug
02080            parcelholder = -1; passparcel;
02090            mapabt = apartmap;
02100            (mapabt + 2) = #1200; ! priority 4 ; reg-set 1
02110            mapsr0 = #000001; ! enable mapping - bit 0
02120          END; ! initmap
02130        END
02140       ELUDOM
```

```
00010       %
00020       ************************************************************************
00030       **********                                                      **********
00040       **********  MODULE : ACCESSTOPCODE                              **********
00050       **********                                                      **********
00060       ************************************************************************%
00070       MODULE commnlib( nolist ) =
00080        BEGIN
00090         BYTE OWN  VECTOR pcode[#20000];
00100
00110         GLOBAL ROUTINE deposit(adr,value) =
00120               pcode[.adr] = .value;
00130
00140         GLOBAL ROUTINE fetch(adr) =
00150               .pcode[.adr];
00160
00170        END
00180       ELUDOM
00190
00200       ;MODULE commnlib( nolist ) =
00210       ; BEGIN
00220       ;  GLOBAL ROUTINE deposit(adr,value) =
00230       ;        .adr = .value;
00240       ;
00250       ;  GLOBAL ROUTINE fetch(adr) =
00260       ;        return ..adr;
00270       ;end
00280       ;eludom
00290
00300
00310
00320
00330       .PDP10
00340       .TITLE COMMNLIB
00350       .CSECT COMM.C
00360
00370       R$0=%0
00380       R$1=%1
00390       R$2=%2
00400       R$3=%3
00410       R$4=%4
00420       R$5=%5
00430       SP=%6
00440       PC=%7
00450
00460
00470              ;DEPOSIT(ADR,VALUE)=
00480       DEPOSIT:
00490              .WORD    22                ;TKM 2 , TO UI
00500              MOV      2(SP),@4(SP)      ;.ADR = .VALUE
00510              .WORD    20                ;TKM 0 , NORMAL
00520              RTS      PC
00530
00540
```

```
00550
00560
00570
00580
00590           ;FETCH(ADR)=
00600   FETCH:
00610           .WORD   23              ;TKM 3   , FROM UI
00620           MOV     @2(SP),R$0      ;RETURN ..ADR
00630           .WORD   20              ;TKM 0   , NORMAL
00640           RTS     PC
00650
00660
00670
00680
00690
00700
00710   COMMNLIB:
00720
00730
00740
00750   .GLOBL  COMMNLIB
00760   .GLOBL  FETCH
00770   .GLOBL  DEPOSIT
00780
00790
00800
00810
00820   .END
00830
00840
00850
```

```
00010       %
00020       ************************************************************************
00030       **********                                                    **********
00040       **********  MODULE : ARITHMETICLIBRARY                        **********
00050       **********                                                    **********
00060       ************************************************************************%
00070       .CSECT ARITH.C
00080       .TITLE ARITH
00090              MQ=177736
00100              R$0=%0
00110              R$1=%1
00120              R$2=%2
00130              R$3=%3
00140              R$4=%4
00150              R$5=%5
00160              SP=%6
00170              PC=%7
00180
00190       MUL:   MOV 2(SP),@#MQ
00200              .WORD   103121
00210              .WORD   4          ;MUL 4(SP),-(SP)
00220              IST     (SP)+
00230              MOV     @#MQ,R$0
00240              RTS PC
00250
00260       MODR:  MOV 4(SP),@#MQ
00270              CLR R$0
00280              .WORD 113107
00290              .WORD 2            ;DIV 2(SP),R$0
00300              RTS PC
00310
00320       DIVR:  MOV 4(SP),@#MQ
00330              CLR R$0
00340              .WORD 113107
00350              .WORD 2            ;DIV 2(SP),R$0
00360              MOV @#MQ,R$0
00370              RTS PC
00380
00390       SHIFT:
00400              JSR     R$1,$SAV2
00410              MOV     #12,R$0
00420              ADD     SP,R$0
00430              MOV     10(SP),R$2
00440              BGE     L$5
00450              NEG     10(SP)
00460              MOV     #1,R$1
00470              BR      L$7
00480       L$6:   ASR     @R$0
00490              INC     R$1
00500       L$7:   CMP     R$1,10(SP)
00510              BLE     L$6
00520              BR      U$2
00530       L$5:   MOV     #1,R$1
00540              BR      L$10
```

```
00550       L$9:    ASL     @R$0
00560               INC     R$1
00570       L$10:   CMP     R$1,R$2
00580               BLE     L$9
00590       U$2:    MOV     @R$0,R$0
00600               RTS     PC
00610
00620       ; ROUTINE SIZE:  28
00630
00640       .GLOBL MUL
00650       .GLOBL DIVR
00660       .GLOBL MODR
00670       .GLOBL SHIFT
00680       .GLOBL $SAV2
00690       .END
```

APPENDIX - 7

DOCUMENTATION OF CHANGES DONE TO THE P-COMPILER

The present version of the P-Compiler called PAS316.PAS is based on the original Pascal-P Compiler developed at ETH Zurich and described in [ ]

The Compiler had been modified to enable efficient interpretation of the P-Code it generates on TDC-316. Major modifications to the P-Compiler are as under:

a) CONST
intsize = 2; realsize = 4; charsize = 2; boolsize = 2;
setsize = 8; ptrsize = 2; lcaftermarkstack = 10;

b) VAR
ppc , maxppc : integer;
(* P-Machine Program Counter. This counter keeps the count of the number of bytes of code generatd, as each P-Code instruction is emitted by the Compiler. It takes into account the assembler generated optimizations also. This counter's value is displayed at the beginning of each line of the listing produced by the compiler; and thus enables the interpreter to point a run-time error correct to one source line of the user program. *)
runtmchk : boolean;
(* the Compiler can now optionally generate code for checking of array index bounds and subrange bounds at run-time. Default : ON *)
errflag : boolean;
(* this flag gets set when an error is detected by the Comiler. *)

c) PROCEDURE
i) insymbol;
(* insymbol has now been split up into three smaller routines. *)
a) inident;
(* accepts lower case alphabets and underscore character *)
b) innumber;
c) insymbol;
(* insymbol now accounts for all printable characters *)
ii) getadch;
(* this routine is newly introduced, getadch is invoked by the compiler during code generation. It checks the attributes of the expression being parsed, and returns the type information in the variable ADCH. *)
iii)gen0 ,gen1 ,gen2 ,load, store, loadaddress;

```
      (* these procedures have been modified to generate the
         appropriate type information with the P-Code mnemonics
         and write the correct PPC value.                    *)
iv)   selector;
      (* array index bounds checks introduced               *)
v)    assignment;
      (* subrange bounds checks introduced                  *)
vi)   casestatement;
      (* the code generated is now CHKS LB UB for range
         checking and the UJC for undefined case labels
         see Fig 2.1                                        *)
vii)  (* All places where Load and Store instructions are gen-
         erated, now if feasible generate the newly introduced
         instructions LDL ,STL and LAL.                     *)
```

APPENDIX - B
****

!PAS316 USER MANUAL
 ****

A user can interact with the PAS316 System through the Command Processor. The Command Processor accepts commands from the TTY, and initiates the various routines necessary to execute the command.

A set of switches have been provided to enable the user to select and deselect some facilities of the PAS316 System. Each occurence of the switch complements the current status of the switch.
for example:
If switch O was ON, then
/O/O -- keeps it on.
/O/O/O -- switches it off.

SWITCHES:
****
L: Listing of the Assembly Language Program, by the Assembler.
O: Display the contents of the file OUTPUT, as they are written by the P-Interpreter on the TTY. This feature is useful for observing the results of a program execution as they are obtained. Especially meant to observe the listfile as it is generated by the compiler on the TTY.
S: Save the Object Code produced by the P-Assembler in a file.
T: Produce the Execution Trace of the P-Instruction execution. Displays on the TTY after every instruction execution the values of the following registers:
Opcode, Program Counter, and the contents of the top 4 words of the Stack.

There are a set of Commands to facilitate the user to run his programs. The commands ask for filenames where needed.

COMMANDS:
****

COPY : Copies one text file into another.
Switches : NONE

TYPE : Types a file on the TTY.
Switches : NONE

PASCAL : Compiles a Pascal program. Files used are:
INPUT - must contain the source program.
OUTPUT- optionally a source listing is produced, and Compiler messages.

```
          PRP   - optionally the P-Code is produced.
          Switches : NONE

PASASM  : Compiles the source program and assembles the
           P-Assembly Language produced by it.
          Switches : L: default = OFF
                     S: default = ON

PASGO   : Compiles and then assembles and executes the
           Pascal program.
          Switches : L: default = OFF
                     O: default = ON
                     S: default = OFF
                     T: default = OFF

ASM     : Assembles the P-Machine's Assembly Language program.
          Switches : L: default = OFF
                     S: default = ON

ASMINT  : Assembles and Interprets the P-Machine's Assembly
           Language program.
          Switches : L: default = OFF
                     O: default = ON
                     S: default = OFF
                     T: default = OFF

EX      : Loads and Interprets the given P-Code.
          Switches : O: default = ON
                     T: default = OFF

KJOB    : Exit from PAS316.
```